tijdschrift van het

nederlands elektronica- en radiogenootschap

deel 54 nr.3 1989

nederlands elektronica- en radiogenootschap

Nederlands Elektronica- en Radiogenootschap
Postbus 39, 2260 AA Leidschendam. Gironummer 94746
t.n.v. Penningmeester NERG, Leidschendam.

HET GENOOTSCHAP

De vereniging stelt zich ten doel het wetenschappelijk onderzoek op het gebied van de elektronica en de informatietransmissie en -verwerking te bevorderen en de verbreiding en toepassing van de verworven kennis te stimuleren.

Het genootschap is lid van de Convention of National Societies of Electrical Engineers of Western Europe (Eurel).

BESTUUR

Ir. J.B.F. Tasche, voorzitter
Ir. H.B. Groen, secretaris
Ir. J. van Egmond, penningmeester
Ir. N.H.G. Baken, programma commissaris
Ir. J.W.M. Bergmans
Ir. R.C. den Dulk
Ir. O.B.M. Pietersen
Ir. P.P.M. van der Zalm

LIDMAATSCHAP

Voor lidmaatschap wende men zich tot de secretaris.
Het lidmaatschap staat open voor academisch gegradueerden en hen, wier kennis of ervaring naar het oordeel van het bestuur een vruchtbaar lidmaatschap mogelijk maakt. De contributie bedraagt *f* 60,– per jaar.

Studenten aan universiteiten en hogescholen komen bij gevorderde studie in aanmerking voor een junior-lidmaatschap, waarbij 50% reductie wordt verleend op de contributie. Op aanvraag kan deze reductie ook aan anderen worden verleend.

HET TIJDSCHRIFT

Het tijdschrift verschijnt zesmaal per jaar. Opgenomen worden artikelen op het gebied van de elektronica en van de telecommunicatie.

Auteurs die publicatie van hun wetenschappelijk werk in het tijdschrift wensen, wordt verzocht in een vroeg stadium contact op te nemen met de voorzitter van de redactiecommissie.

De teksten moeten, getypt op door de redactie verstrekte tekstbladen, geheel persklaar voor de offsetdruk worden ingezonden.

Toestemming tot overnemen van artikelen of delen daarvan kan uitsluitend worden gegeven door de redactiecommissie. Alle rechten worden voorbehouden.

De abonnementsprijs van het tijdschrift bedraagt *f* 60,–. Aan leden wordt het tijdschrift kosteloos toegestuurd.

Tarieven en verdere inlichtingen over advertenties worden op aanvrage verstrekt door de voorzitter van de redactiecommissie.

REDACTIECOMMISSIE

Ir. M. Steffelaar, voorzitter
Ir. C.M. Huizer

ONDERWIJSCOMMISSIE

Prof. Dr. Ir. W.M.G. van Bokhoven
Ir. R. Brouwer
Ir. J. Dijk

PARALLEL COMPUTING AT PHILIPS:
The Object-Oriented Approach

Ir. Hans Oerlemans, Ir. Eddy Odijk, Wim Bronnenberg
Philips Research Laboratories Eindhoven

Symbolic applications pose challenging requirements on parallel computer architectures and the design and expression of parallel algorithms. In this paper, an object-oriented approach, to meet these requirements is described. It treats the research and design performed at Philips Research Laboratories Eindhoven in the framework of the current parallel processing projects. In the Parallel Object-Oriented Language POOL, a program is subdivided into a large number of so-called objects, which communicate by sending messages. The language offers explicit parallelism and support for structuring large applications to be executed on POOMA, the Parallel Object-Oriented MAchine. The POOMA architecture consists of a collection of self contained computers, comprising a CPU, local memory and a communication unit to connect these computers via a point-to-point packet switching network.

1 Introduction

While many of the efforts in parallel computers for numeric purposes emphasize compatibility with existing programming languages, new ways have to be followed for symbolic parallel computers.

In a language-first approach, a number of choices can be made when it comes to programming such systems. For instance, the models of functional and logic programming allow parallel execution, where, in principle, the parallelism is invisible to the programmer and is extracted by the implementation (implicit parallelism).

Central themes in parallel computer architecture are formed by the communication structure and processor support for new language features. For example, several shared memory systems have been proposed with busses or multi-staged switching networks as communication means.

This paper presents a survey of a different approach to parallel computing: an *object-oriented* approach. It presents the concepts of the Parallel Object-Oriented Language **POOL** and a highly parallel, general purpose computer system for execution of programs in this language: the Parallel Object-Oriented MAchine, **POOMA**. In POOL, a program is subdivided into a large number of so-called objects, which communicate by sending messages. The POOMA architecture contains many identical computers that are connected by means of a packet-switching network. Each computer supports the execution of a number of POOL-objects.

The approach described here, forms the basis of a number of parallel processing projects being undertaken at Philips Research Laboratories Eindhoven. These projects are:

- DOOM (Decentralized Object-Oriented Machine), [Bronnen87]
  In DOOM an effort into the construction of a general purpose parallel programming system is being undertaken. DOOM is executed as subproject A of Esprit project 415.
- PRISMA (Parallel Inference and Storage MAchine), [Apers88] The main aim of PRISMA is to adopt parallel object-oriented programming techniques for the construction of applications manipulating large data and knowledge bases. PRISMA is partially supported by the Dutch "Stimuleringsprojectteam Informaticaonderzoek" (SPIN).
- TROPICS (TRansparent Object-oriented Parallel Information Computing System)
  TROPICS is a joint effort of four major computer manufacturers (Philips, Nixdorf, Olivetti, Thomson) to introduce parallel object oriented systems as the basic platform into the office automation environment. The TROPICS project is supported by Esprit (project 2427).

The main issues to be addressed in these projects, cover a wide spectrum to obtain a well balanced design:

- The construction of prototypes of the Parallel Object-Oriented MAchine, **POOMA**, consisting of identical self-contained computers, each having a CPU, local memory, disc and communications means, which are connected in a direct packet-switching network. Each computer, called a node of the system, has a copy of the operating system kernel. This kernel performs local resource management, and cooperates with

the other kernels for global operating system tasks. For DOOM and PRISMA prototypes containing some 10 nodes are realized, for TROPICS prototypes containing 100 nodes are under construction.

The prototype systems are all connected, as a satellite, to a host computer, where the programming environment resides. This setup was chosen for the prototypes to postpone the programming effort on an environment, to a development stage.

- A parallel object-oriented language, **POOL**, in which significant application programs can be programmed. The language provides the user with control of parallelism and granularity. It is important for the language to have clear semantics. Support for the verification of programs is desirable, and becomes even more important in a parallel environment. The design of a proof system forms therefore part of the effort.

- Construction of applications in the area of symbolic processing that assess the potentials of POOL and the POOMA system architecture. The first of these is a parallel database management system. The second is a parallel version of the analytical component of the Rosetta natural language translation system [Landsb87]. Rosetta is currently being designed in the same department at Philips Research.
  In the framework of TROPICS a multimedia document systems and a carthographic database systems are being developed.
  While our emphasis is on symbolic processing, the VLSI circuit simulator of AEG [Mehring87] and applications designed by others will demonstrate the applicability of POOMA for numerical applications.

From the above it follows that the new issues for the design of programming languages, application programs and computer systems, raised by the quest for massive parallelism, are approached in an integral and coherent manner. In our view this is essential to successfully address all of the above issues.

2 The programming language POOL

POOL is an acronym for "Parallel Object-Oriented Language", and denotes a family of languages designed in the course of the parallel processing projects at Philips Research.

The effort in language design was directed at a language that exploits the ideas of object-oriented programming in order to make the programming of systems with a large amount of parallelism feasible. It was not meant as a tool for rapid prototyping (where the excellence of many existing object-oriented languages lies), but as a language to program large, well-designed systems, with as many facilities for static checking as possible.

2.1 Introduction to POOL

The essence of object-oriented programming is the subdivision of a system into *objects*, which are integrated units of data and procedures. Objects are entities of a very dynamic nature: they can be created dynamically, the data they contain can be modified and they even have an internal activity of their own, as will be described later.

The most important property of objects is that they form the units of abstraction and protection. An object can have *variables* (also called instance variables) to store its internal data in. A variable can contain (a reference to) an object. Changing (assigning to) a variable causes it to refer to a different object than before. The variables of an object cannot be accessed directly by other objects: there is a a clear separation between the inside and outside of an object.

Objects may only interact by sending *messages* to each other. Each object states explicitly to which object it *sends* a certain message. Upon accepting a message, the object will execute one of its procedures (indicated by the sender) and the object that is the result of this execution will be sent back to the sender. In object-oriented languages such a procedure is called a *method.* Note that the method executed by the object can access its variables. As the object has explicit control of its interaction with other objects, it can keep its internal data in a consistent state.

Objects are entities of a semantic nature. On the syntactic level the corresponding notion is that of a **class**. A class is a description of the behaviour of a set of objects, its instances. It describes the number and kind of their variables, and the methods that are executed in response to messages. A number of so-called *standard classes* (e.g., Integer and Character) are already predefined.

Beside the methods attached to each instance of a class, there are also *routines*: procedures that are related to a class rather than to a specific object. They can be called by all instances of all classes in a program. A typical task for a routine is to create and initialize new objects of its class.

POOL has a strong typing mechanism: with each variable a class is associated (its *type*); it may contain references to objects of that class only. With

every expression in the language, a class (again called its type) can be associated statically. A program is only considered valid if for every expression its type is the same as required by its context. Thus, many programming errors can be detected before the program is executed.

2.2 Parallelism in POOL

There are several ways to combine object-oriented languages with parallelism. In Smalltalk-80, [Goldberg83] the traditional concept of a process is introduced, where several of these may execute in parallel, each acting in the same way as if it were an ordinary sequential object-oriented program. The same additional constructs are necessary as with traditional processes to get the concurrency under control. For example, in Smalltalk-80, synchronization and mutual exclusion is done by semaphores.

A better approach to integration starts by associating a process with every object. By doing this, we also get a very natural model for a purely sequential execution, in which at any time only one object is active, by adhering to the following restrictions: (1) execution starts with only one active object, (2) the sender of a message always waits until the corresponding method has returned its result and (3) an object is only active when executing methods.

Starting from such a sequential execution there are several possibilities to allow genuine parallelism, each one characterized by which of the above restrictions is relaxed.:

(1) By allowing to start execution with several active objects, we obtain a static amount of parallelism.

(2) If we allow the sender of a message to go on with its own activities without waiting for the result of the method, the receiver can start executing in parallel with the sender *asynchronous message passing.* By creating more objects and sending them messages, the number of concurrently active processes can be increased quickly. This principle is employed, for example, in the actor languages developed at MIT [Hewitt77]. (3) Another possibility is to specify for each object a *body*, an activity of its own, which it executes without the need for a message to initiate it. In this case the moments when the object is willing to accept, or *answer* a message must be indicated explicitly. By selectively indicating which methods it is willing to execute, an object can get complete control over the consistency of its data. Because objects have a body, the concurrency need not come from the concurrent execution of a message sender and the processing of the message by the receiver, so in this situation synchronous message passing would be sufficient.

POOL offers both synchronous and asynchronous communication as well as the possibility to start execution with several active objects. A rationale for this and other choices has been presented in [America87].

2.3 Resulting characteristics of POOL

The object-oriented approach acknowledges the fact that the programmer is responsible for and has the best knowledge of the detection and use of parallelism in his application. To relieve his task, object-oriented languages supply the user with a natural method for structuring and partitioning combined with a message-passing mechanism for communication and synchronization.

In POOL every data item, from a boolean to a complete database, is represented by an object. All these objects have the same general way of interacting with each other. An important benefit of this unification is the resulting simplification in the language.

The dynamic structures exhibited by many symbolic applications, for which the fifth generation parallel architectures are targeted, are matched in object-oriented languages by the mechanisms for dynamic creation of objects and for communication to other objects.

The subdivision of a system into objects yields coinciding notions for both information hiding and protection (essential for reliable and maintainable software) and concurrent execution. Security is further enhanced by the strong typing mechanism. A *unit* mechanism, not described here, is an important feature to further enhance abstraction and to improve reusability of program parts [America88], [America88a].

3 The POOMA system

POOMA is an experimental system in which the exploration of parallelism in object-oriented programming and the principles of efficient implementations are investigated.

We start describing the implementation by having a closer look at the computational model of POOL. As has been shown in the previous section, a POOL program in execution consists of processes (objects) which internally show sequential control-flow, or imperative, behaviour. These processes can be dynamically created during program execution upon request by other objects. Explicit deletion of objects is not possible. Communication and synchronization between objects in POOL are performed by (a)synchronous message passing. A sender can only send messages to objects to which it has a reference in one of its variables. If more than one message is

sent to an object (by different sending objects), the first received message corresponding to a method in the answer statement will be invoked.

Obviously, this computational model does not incorporate any notion of sharing data. In the extreme, one could envisage an implementation consisting of a large number of self-contained computers, i.e consisting of a processor and local memory, each of which executes a single object. The computers would then be connected through some, virtually fully connected, network that passes messages between objects. Although such an implementation is not realistic in terms of cost-performance, it provides an interesting abstract machine model. Using such an Abstract POOL Machine (APM) model, one can show the various system components and design decisions that are required to map the dynamic, and changing, graph of communicating objects onto a static graph of POOMA nodes. The mapping of processes occurs dynamically, and the communication can take place between objects anywhere in the system, allowing a many-to-one communication pattern.

The POOMA system architecture maintains the concept of self-contained computers, connected by a network. It can be considered as a restricted version of the APM sketched above. In the sequel the present prototype architecture and the operating system for POOMA will be described. The "design space" and required manpower have been restricted to the essence of the project goals and a manageable size. Therefore, a number of issues is excluded from the first design. The system design was focussed at the two basic aspects of architectural support: the object execution within a node, and the communication between nodes.

3.1 The POOMA architecture

In this section the architecture of a POOMA node is presented, followed by an introduction of the network structure that connects these nodes. A node in the prototype system basically contains a data processor, a memory subsystem, and a communication processor.

The *data processor* (DP) executes the code of the objects which reside on the node. Unlike a node in the APM, a POOMA node is able to execute many (presumably some 10 to 1000) objects. The processor architecture must therefore support multiprocessing. Efficient process switching is a first requirement. Most presently available microprocessors are designed to support processes of coarse granularity and have not been optimized towards this aspect. For the prototype system, where the Motorola 68020 and SPARC microprocessors will be used, this may lead to some performance degradation.

In an accompanying study, we have shown that feeding independent instruction streams through a pipelined processor implementation, is in harmony with the POOMA and RISC concepts and removes bypasses and interlocks from the implementation, see [Bronnen86].

The *memory* must host the operating system and accommodate for the code, the stacks and the message queues of the residing objects. As will discussed in the section on the OS, a paged virtual memory scheme has been adopted to support memory management.

The last component is the *communication processor*. In order to avoid that interprocessor communication becomes the new bottleneck in parallel systems, hardware support is appropriate. We have taken it to the point where data packets (of fixed 256 bit size), after transfer by the DP to the local CP, are injected by that CP into the network, passed on by other CPs to the destination node and are handed over to the DP there. Unlike with cut-through routing, packets are buffered as a whole before they are passed on by a CP. The design of the CP is such that it guarantees deadlock and starvation free routing of packets, independently of network topology or size. Furthermore it incorporates free routing, i.e. packets may be forwarded to the same destination via different routes while maintaining efficient usage and administration of buffer space. The algorithm used in the POOMA CP for this purpose, is called class climbing [Annot87],

The routing facility offered by the CP provides a powerful and efficient mechanism for higher layers of the system, and avoids interruption of the intermediate DPs. A set of (FIFO) buffers in between CP and DP serve the purpose of decoupling the production and the consumption rates of messages in both directions.

Our main goal is to develop, for reasons of cost and performance, a one chip VLSI implementation. Simulated performance of an implementation with 4 bidirectional links, used in a network with average distance of 4, is 75000 packets/s which is very close to the available bandwidth at a 20Mb/s data-rate. Currently working is an implementation of the CP using commercial available components. Its performance is 50000 packets/sec. This breadboard CP is used in our prototypes.

Another concern of the architecture is the design of a topology for the communication network [Odijk87]. For the topology many candidates with different characteristics exist. In general purpose machines one prefers one that suits the general case where the communication patterns are of an irregular nature and may vary during execution.

Given a maximum number of links (the degree of a network) that can be afforded with the implementations' technology, criteria for selection are obtained by the diameter and the regularity of the network, the extensibility and its tolerance to defective nodes or links. Further criteria may be derived from the number of alternative paths between nodes and the ease of routing or from physical properties of the links, such as the width and transmission speed.

The main interests in the POOMA design were to find a good ratio between the number of links and the diameter and have a high degree of regularity. Considering the importance of regularity of the network, we have proposed to build networks as cartesian products (cubes) with optimal chordal rings as the elementary building blocks. The resulting generalized chordal ring cubes yield a lower degree and diameter than toruses and hypercubes and have excellent fault tolerance properties. The choice of the exact topology for POOMA can be postponed, thanks to the flexibility of the communication processor.

3.2 The POOMA Operating System

The main target for the OS of the POOMA prototype is: efficient execution of POOL programs. Therefore facilities one also needs like a programming environment, etc. are not offered by the POOMA OS itself, but by the OS of the host computer that POOMA is connected to.

The remaining task of the OS deals with several aspects, such as (local and global) resource management, process management and communication handling. The latter is described in [Haan88]. A number of these issues that are kind of special for the POOL environment, is described below.

Allocation of objects onto nodes

In general, when an object is created, the object should be "added" to a node where other objects already reside. The selection of the node must be done dynamically, due to the creation of objects during program execution, and should aim at an optimal use of resources. Conflicting criteria are: 1) processor load, 2) memory occupation, and 3) communication load. According to the first, objects which can operate in parallel should be placed on different nodes. The third leads to keeping objects on the same node when they are interacting.

POOL programs can be annotated with pragmas indicating a set of nodes from which the OS can choose for allocation. On basis of this information and by means of the gathered load balancing information indicated above, a "proposed allocation node" is calculated. A request for allocation is then sent to that node, together with some information about the object to be created (e.g., its code-size). If the request is not accepted, the OS has to calculate a next candidate.

Garbage Collection

In POOL, objects are never destroyed explicitly. However, when an object is only waiting for messages or its own activity has terminated, and furthermore no other object in the system has a reference to it, then it may be safely removed from the system. Because of the possibly cyclic reference structure, the basic garbage collection algorithm (GC) that is needed should be Mark and Sweep. For a proper working of the GC it is necessary that references to objects (e.g. in variables) are traceable. There are several ways to accomplish this. One way is by means of a tagged memory, another way is using two stacks per object: one for references to objects and one for other data.

Most traditional mark and sweep garbage collectors assume that no other processing is going on during a GC-phase. This is not acceptable for a multi-processor system, because at each moment only a fraction of the processors will be actually involved in GC and most processors would thus be idle. However, special care has to be taken in order to let GC go on in parallel with other processing. A special synchronization problem that comes up in a decentralized or distributed system is how to detect that a mark phase or sweep phase can start or has finished. A more detailed description can be found in [Aug87].

Memory management

The memory of a POOMA node will be used to store three different types of items:
Stacks: for each object residing on the node a dynamically extending and shrinking stack is needed.
Message queues: for each method of each object residing on the node a dynamically extending and shrinking FIFO message queue is needed.
Code blocks: in principle one code block per object is needed. However objects of the same class will share code blocks.
Obviously, these types of items exhibit quite different logical behaviours. More, their sizes differ and vary per object. Therefore a static split-up of a node's memory is out of the question.

For POOMA a two level memory management system has been designed, with a hardware supported paging system as the lower level. This paging system will *not* be used for extending the memory by means of a background store (like in most operating systems), but solely to obtain the desired flexibility of dynamic allocation of storage blocks. It supports the OS with sufficient virtually contiguous memory space which is split up in fixed size segments. In

this way, e.g., one segment can be reserved per datastack. Increases and decreases in stack size, requiring allocation and deallocation of (physical) pageframes can then be handled independently from the behaviour of other segments.

3.3 State of affairs

Various prototypes have been realized. Based on the following node architecture:
A processor-memory subsystem being composed of a PG2100 processor board, [PG2100] plus an extra 12 MByte memory extension board. The PG2100 is based on the Motorola 68020 microprocessor and has an accompanying memory management unit (68851) and floating point unit (68881). Furthermore 4 Mbyte of on board memory; together with the memory extension board each node is equipped with 16 Mbyte. In its prototype version (bread-boarded) the communication processor takes two (extended double Euro) boards. Each node thus consists of 4 boards.

For DOOM a prototype consisting of 12 nodes according to the above architecture has been realized. It supports a prototype implementation of POOL2, [America88].

For PRISMA a prototype consisting of 8 nodes has been realized. Each node consists of the components described above plus a 100 MByte disc (for supporting the database application). It supports a prototype implementation of POOL-X, [America88a].

For TROPICS a number of prototypes are envisaged. A first prototype has been realized that consists of 100 nodes according to the above description and extended with 50 discs of 300 MByte each. For experimental purposes this prototype is also equipped with a component that allows the topology of the network to be changed under software control. This allows for easy and flexible experimenting with different network configurations.
A second prototype will be based on the SPARC microprocessor. It will be equipped with some 32 MByte of main memory per node and 50 discs. For the communication processor a VLSI relisation will be used. Using this technology the number of boards for each processing node will be reduced to one.

To allow the application programmers to obtain experience with the POOL language, implementations for executing POOL on sequential systems have been realized (an interpreter and a compiler plus run-time system running on SUN - UNIX). The interpreter package incorporates a tracing and debugging environment. A third package, emulates the the POOMA execution of POOL programs on the sequential (SUN - UNIX) system. This interpreter configures a network of POOMA nodes, according to user specification, and executes POOL programs on this network, while administering progress. Costs of instruction execution and communication have been derived from the actual POOMA design. Using the latter package, characteristics are being measured of early versions of the applications mentioned in the introduction.

References

[America87] Pierre America: *POOL-T - A parallel object-oriented language*, in: A. Yonezawa, M. Tokoro, eds.: Object-Oriented Concurrent Programming. MIT Press, 1987, pp. 199-220.

[America88] Pierre America: *Definition of POOL2, a parallel object-oriented language*, Doc. No. 364, ESPRIT Project 415A, Philips Research Laboratories, Eindhoven, the Netherlands, 1988.

[America88a] Pierre America: *Language definition of POOL-X*, Doc. No. 350, PRISMA Project, Philips Research Laboratories, Eindhoven, the Netherlands, 1988.

[Annot87] J.K. Annot, R.A.H. van Twist: *A Novel Deadlock free and Starvation free Packet Switching Communication Processor*, Proceedings of the PARLE Conference, Springer Lecture Notes in Computer Science, Vol 258 (I), pp 68-85, 1987.

[Apers88] P.M.G. Apers (UT), M.L. Kersten (CMCS), A.C.M. Oerlemans (PRLE): *PRISMA Database Machine: A Distributed Main-Memory Approach*, Proceedings of the Int. Conf. on Extending Database Technology, Italy 1988.

[Aug87] Lex Augusteijn: *Garbage Collection in a Distributed Environment*, Proceedings of the PARLE Conference, Springer Lecture Notes in Computer Science, Vol 259 (II), pp 75-93, 1987.

[Bronnen86] W.J.H.J. Bronnenberg, M.D. Janssens, E.A.M. Odijk, R.A.H. van Twist, *The Architecture of DOOM*, Proceedings of the ESPRIT-415 Summerschool 1986, in: Springer Lecture Notes in Computer Science, pp. 227-269, 1987.

[Bronnen87] W.J.H.J. Bronnenberg, A.J. Nijman, E.A.M. Odijk, R.A.H. van Twist, *DOOM: A Decentralized Object-Oriented Machine*, IEEE Micro, October 1987, pp. 52-69.

[Goldberg83] A. Goldberg and D. Robson: *Smalltalk-80, The Language and its Implementation*, Addison-Wesley 1983.

[Haan88] P. den Haan, F. Hopmans, *Efficient Message Passing in Parallel Systems with Limited Memory*, Proceedings of CONPAR 88, United Kingdom.

[Hewitt77] C. Hewitt: *Viewing Control Structures as Patterns of Message Passing*, Artificial Intelligence, Vol. 8, 1977, pp. 323-364.

[Landsb87] S.P.J. Landsbergen: *Isomorphic Grammars and their use in the Rosetta Translation System*, Paper presented at the Tutorial on Machine Translation, Lugano 1984, in: M. King (ed), Machine Translation the state of the art, Edinburg University Press, 1987.

[Mehring87] P. Mehring and E. Aposporides: *Multilevel Simulator for VLSI*, Proceedings of the PARLE Conference,Springer Lecture Notes in Computer Science, Vol 258 (I), pp 446-460, 1987.

[Odijk87] E.A.M. Odijk, R.A.H. van Twist: *Networks for Parallel Computer Systems*, Proceedings of the Compeuro '87 conference, pp 779-782, IEEE 1987.

[PG2100] *PG2100 VMEbus 32-bit single board computer*, Users manual, Philips Export B.V. 1988, Eindhoven.

Voordracht gehouden tijdens de 369e werkvergadering.

NEDERLANDS ELEKTRONICA- EN RADIOGENOOTSCHAP
(365ste werkvergadering)

UITNODIGING

voor een lezingen**avond** op **dinsdag 31 januari 1989 in het 3e hoofdgebouw van de N.V. Nederlandse Spoorwegen te Utrecht.**
THEMA: TELEKOMMUNIKATIE BIJ DE NEDERLANDSE SPOORWEGEN.

PROGRAMMA

vanaf 19.45 uur	Koffie.
19.50 - 20.00 uur	Ontvangst door **IR. P. F. A. M. OTTEN**, Chef Afdeling Elektrotechniek, Dienst van Infrastruktuur.
20.00 - 20.45 uur	**IR. A. V. P. VAN DER LINDEN**, ontwikkeling en beleid telematika; VERNIEUWING VAN DE TELEKOMMUNIKATIE-INFRASTRUKTUUR VAN DE NEDERLANDSE SPOORWEGEN.
20.45 - 21.00 uur	Videofilm: ELEKTROTECHNIEK ACHTER HET SPOOR.
21.00 - 21.45 uur	**IR. R. S. DE HAAS**, chef sektor ontwikkeling elektrotechniek; NIEUWE TELEMATIKA-TOEPASSINGEN BIJ DE NS.
21.45 - 22.00 uur	Diskussie.

Aanmelding voor de lezingenavond dient te geschieden vóór 23 januari 1989 door middel van de aangehechte kaart, **gefrankeerd** met een **postzegel** van **55 cent**. Niet leden dienen een entreeprijs van *f* 15,00 te betalen.

Het 3e hoofdgebouw van de NS ligt aan de stadszijde van het station Utrecht CS op loopafstand. U loopt vanuit de stationshal naar het streekbusstation en volgt dan het railspoor van de sneltram tot het einde . De zijingang van het 3e hoofdgebouw (groot donker bakstenen gebouw) bevindt zich dan schuin links voor u.

Namens het NERG bestuur,
Ir. N. H. G. Baken,
Tel. 070 - 436482.

Voorburg, januari 1989.

VISUALIZATION OF 3-D EMPIRICAL DATA: THE VOXEL PROCESSOR

ir. Peter L.J. van Lieshout, ir. Wim Huiskamp

TNO Physics and Electronics Laboratory, P.O. box 96864, 2509 JG The Hague, The Netherlands

Image processing is one of the areas which are known to be very computational intensive. To achieve interactive response of imageprocessing systems usually dedicated systems or (mini-) supercomputers are necessary. Imageprocessing is also an area in which the application of parallelism is very suitable, because of the large amounts of (similar) data involved. In this particular case, 3-D images (images in 'voxel space') have to be processed interactively. Operations on the voxel space involve 3-D image processing algorithms and visualization of the data from arbitrary viewing angles, and with several options for the type of rendering. This paper describes an alternative for special hardware or supercomputers for a voxel processor, based on a network of INMOS T800 Transputers.

Introduction

The TNO Physics and Electronics Laboratory (TNO-FEL) in the Hague is a part of the TNO Division of National Defence Research (TNO-HDO).

The activities of TNO- FEL focus primarily on operational research, information processing, communication and sensor systems. To support the fast data-processing usually required in sensor systems applications, research was started into parallel processing techniques. This research has now resulted in three major application areas : radar data processing, real-time computer generated imagery and 3D image analysis, processing and visualization. 3D image processing and visualization is the subject of this paper. With the growing availability of 3D scanning devices, the need for high performance processing and display systems increased significantly. The development of an experimental parallel processing system is described for the visualization of three dimensional voxel based images. The aim is the visualization of the (unknown) object in such a way that its spatial structure can be understood. An additional demand is that the system is fast enough to be used interactively. Because of the large number of voxels involved, a considerable processing capacity is required. Processing the data in parallel on a network of Transputers provides the necessary computing power. The volume data may be visualized in several ways, involving operations like object transformation, hidden-surface removal, depth-shading and cross-sectioning. The main advantages of the proposed architecture over dedicated hardware solutions are:

- cost/performance ratio
- flexibility
- expandability

The Voxel space

Volume images are normally represented as a series of parallel two dimensional slices. These slices may have been obtained from several possible sensor systems, examples are Computer
Tomographic- (CT), Nuclear Magnetic Resonance- (NMR), Ultra Sounding or Optical- (LASER) scanners. Voxel representations are very suitable for applications with 3D empirical data. However synthetic data may also be used, examples are: solid modeling and fluid dynamics simulations. Before volume rendering became feasible, experts had to interpret every single slice to deduce the 3D information. Until recently, computer assisted techniques to visualize the volumes were based on displaying contours only, because of the processing time involved. These contours often had to be traced manually from the actual data. Full use of the 3D data could only be made through off-line computing.

Several architectures based on dedicated hardware have been proposed to increase performance (Ref. 1, 2).Dedicated systems however have the disadvantage of inflexiblity to any change in rendering options or object sizes (also the costs are high). Other systems make use of high performance general purpose machines, which are always very expensive and not always suitable. This explains the reason for TNO-FEL to apply a system of programmable (low cost) processors operating in parallel. Prototype data for the voxel processor was obtained from an experimental Confocal LASER Scanning Microscope (CLSM). The CLSM can be focussed on several consecutive layers of the object, producing a slice of data for each layer. A slice typically consists of 256256 volume elements (voxels), with an intensity resolution of 8 bits per voxel (FIG. 4). The number of slices may vary, but a typical value is 32 to 256 (FIG. 2). This data structure is called the voxel space. Examples of application areas for the CLSM are medical-and biological-research and inspection (e.g. Integrated Circuits). Voxel space sizes depend largely on the sensor type, in CT scans for example it is possible to get resolutions of 512*512*128 with 12 bits per voxel, and these numbers still grow. The TNO-FEL voxel processor has the modularity to deal with varying size-and performance-demands.

The 3-D Reconstruction

The Voxel space consists of a block in 3-D space (FIG. 3). Displaying this data under different angles on a 2D screen involves a 3-D transformation of the object space into the display space (FIG. 4). Basically such a transformation consists of a vector-matrix multiplication on each voxel coordinate (i.e. a vector). The matrices for a rotation around the X-, Y-, and Z - axis with rotation angles A,B and C respectively are :

$$R_x = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos A & -\sin A \\ 0 & \sin A & \cos A \end{vmatrix}$$

$$R_y = \begin{vmatrix} \cos B & 0 & \sin B \\ 0 & 1 & 0 \\ -\sin B & 0 & \cos B \end{vmatrix}$$

$$R_z = \begin{vmatrix} \cos C & -\sin C & 0 \\ \sin C & \cos C & 1 \\ 0 & 0 & 1 \end{vmatrix}$$

These matrices are concatenated into a single matrix before the actual multiplication:

$R = R_Z * R_Y * R_X =$

$$\begin{vmatrix} (cB*cC) & (sA*sB*cC-cB*sC) & (cA*sB*cC+sA*sC) \\ (cB*cC) & (sA*sB*sC+cB*sC) & (cA*sB*sC-sA*cC) \\ -sB & (sA*cB) & (cA*cB) \end{vmatrix}$$

('c' for cos, 's' for sin).

Suppose a resolution of 128 is used in X-,Y- and Z-direction with 1 byte per voxel. The voxel space then occupies 2 Megabytes. 2 Million vector-matrix multiplications of the kind described have to be performed to compute the correct orientation. Every vector-matrix multiplication consists of 9 multiplications and 6 additions. To compute a new projection therefore 18 million Mults and 12 million Adds would be needed. And this is just the pure computational load, without any overhead like instruction

fetches etc. Fortunately, there are possibilities for a simplification. Since vector-matrix multiplication is a linear operation and basically all of the voxel coordinates have to be transformed, it is not necessary to perform this multiplication for each and every coordinate. We may instead use previously calculated results to compute the next. In that case three simple additions are needed to step from one transformed coordinate to the next. This method offers a considerable reduction in the computational load.

The first step is to transform the unit-vectors from object-space into display-space, by multiplication with the previously calculated rotation matrix R.

$$\begin{vmatrix} nx.x' \\ nx.y' \\ nx.z' \end{vmatrix} = \begin{vmatrix} 1 \\ 0 \\ 0 \end{vmatrix} * R$$

$$\begin{vmatrix} ny.x' \\ ny.y' \\ ny.z' \end{vmatrix} = \begin{vmatrix} 0 \\ 1 \\ 0 \end{vmatrix} * R$$

$$\begin{vmatrix} nz.x' \\ nz.y' \\ nz.z' \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \\ 1 \end{vmatrix} * R$$

The transformed coordinates (x',y',z') of voxel (x,y,z) are now found with :

$$\begin{vmatrix} x' \\ y' \\ z' \end{vmatrix} = x* \begin{vmatrix} nx.x' \\ nx.y' \\ nx.z' \end{vmatrix} + y* \begin{vmatrix} ny.x' \\ ny.y' \\ ny.z' \end{vmatrix} + z* \begin{vmatrix} nz.x' \\ nz.y' \\ nz.x' \end{vmatrix}$$

By simply changing only 1 dimension at the time (i.e. moving along the X-, Y-or Z-axis) a new coordinate is now generated by just three additions. The projection of the 3-D data onto a 2D surface (the screen) involves the hidden surface elimination: 'distant' voxels are obscured by 'closer' voxels if they are projected on the same location on the screen. Comparing the z-value of a new pixel with the z-value of the pixel already present on that screen location (Z-buffer algorithm), is avoided by traversing the voxel space in a back-to-front direction. When generating the screen this way, new pixels can simply overwrite any old value (Painters algorithm).

Several ways of rendering the transformed data on the screen are possible, the available options are:

a) Display the object's intensity, as seen from the selected orientation ('front view'). (Photo 1)

b) Display the object's 'distance' from the screen at each location, resulting in a realistic depth illusion. ('depth shading').

c) Display the object's density at each screen location, ('integrate function').

d) Display an intensity related to the layer from which the visible voxel originated. ('layer view'). (Photo 2)

e) Select a 'Volume-Of-Interest' within the available voxel space (this volume must be block-shaped). Through this option uninteresting or disturbing parts of the voxel image may be 'peeled away'. (Photo 3)

f) Select a cutting plane through the object; voxels in front of this plane will not be visualized. This option will create a cross-section through the object after rotation. (Photo 4)

g) Select a threshold; voxels with a value below this threshold will become transparent.

h) Edit and select different colour look-up tables. This feature enables the use of pseudo-colours or grey-scale transforms for certain intensity values, thereby increasing the visibility of interesting areas.

The original images from the scanning device tend to be noisy in many cases, so noise filters are needed. Further image analysis operations (edge detectors etc.) are also provided. Currently implemented 3D image processing algorithms are :

- Mean filter.
- Sobel and Roberts edge detectors.
- Laplace filter.
- Median filter.

These filters are based on their 2D counterparts and operate on a (3*3*3) space.

Parallel Processing.

Computer applications tend to need increasing amounts of processing capacities. Single processor systems are reaching the limits of performance improvements. It is obvious that using more processors running in parallel should provide (theoretically) unlimited power. Many existing multi-processor systems use a common communications channel (the bus) for interconnections. With a growing number of processors the bus capacity becomes a bottleneck for system performance. Communication bandwidth of the network should be increased also when processors are added. Providing processors with direct (point to point) connections for all data exchange will supply this increased bandwidth.

Several classes of multi-processor systems may be defined. A common way to distinguish classes is between SIMD (Single Instruction stream Multiple Data stream) and MIMD (Multiple Instruction stream Multiple Data stream) type parallelism. In SIMD parallelism each processor in the network will execute the same instruction (synchronously) on different data. Array processors fall in this class. Examples are image processing applications where each processor performs the same filter operation on a different part of the image. When using MIMD parallelism processors can all be running different programs, possibly sending results to others when they are finished. Examples are pipelined systems or multi-user applications.

Many existing sequential programs could benefit from being able to perform more than one action at a time. It is however generally not trivial to implement a parallel program on a processor network. Problems arising are:

- Decomposing the problem in a number of processes running in parallel.
- Allocate processes to processors and select the network topology.
- Load-balancing the processors.
- Distributing data across the processors.
- Efficient inter-processor communication.
- Synchronization between processors.
- Debugging the software.

The INMOS T800 Transputer is a computer-on-a-chip, containing a 32 bit RISC ALU, a 64 bit Floating-Point Unit, memory and four high-speed (1.5 MByte/s) input/output links for point-to-point communication (Fig. 8, Ref. 3). The Transputer was specifically designed for efficient parallel processing : it is a high performance component (10MIPS, 1.5 MFLOPS), with an on-chip process scheduler and low-overhead communication facilities. A network of Transputers may be constructed by connecting them via links. Each Transputer in a network has (private) local memory to store program and data. Transputers
may be programmed in high level languages like PASCAL, FORTRAN or C. These languages must have facilities added to implement the special features of the Transputer (processes running in parallel, communication etc.). OCCAM is a language that was created by INMOS to describe parallel processing and communication via channels (Ref. 4). In fact the Transputer may be considered a hardware implementation of OCCAM.
Transputer versions without the floating-point unit are also available : the T414 (32 bit) and the T212 (16 bit).

Transputer networks belong to the MIMD class of parallel processing systems, all nodes in a network are basically independent units, communicating and synchronizing only
when necessary. A MIMD network is the most flexible solution to parallel processing, since part of the network may actually be operating in SIMD-mode . At TNO-FEL, research has concentrated on the Transputer as the computational element in parallel processing applications, because of its useful features, high performance and software support.

Parallelism may be accomplished in 3D image processing by splitting up the computations in either display space or in object space :
Display space parallelism implies that each processor is assigned to a certain area of the final image (e.g. a number of scanlines). Since views of the rotated voxel image will be generated, this solution means that each processor must have access to the complete voxel space.
Complete access is possible when a voxel image copy is stored in each processor (large memory requirement) or alternatively, processors could request voxel data elements when needed from a central store (communication overhead). Load-balancing may be a problem, since the most computation intensive parts in the display-space will shift according to the rotation angle. Ray-tracing is a typical example where parallel processing in display space is often used. The load-balancing can be tackled by implementing a processor farm construction. In this construction a controller process "farms out" a new piece of work (i.e. a part of the display) to each processor in the network as soon as this has finished work on a previous part. The controller does not need to know which processor will actually perform the job. Object space parallelism is based on access of a limited part of the original voxel image. This implies that each node is assigned to a section of the voxel image, which is stored locally. A node will produce the contribution of the local data to the result. The actual result will be available after combining (merging) all the contributions. The advantages of this method over the previous one are :

- Less memory requirement.
- Fast access to the (local) voxel data.
- Good load-balancing, all contributions will need the same computation time, when the voxel data sizes are equal.

Disadvantages are : the overhead of the merging operation and the fact that some data calculated by the nodes may not be needed in the final result.

The advantages of the second method where strong enough to use object space parallelism in the voxel processor system.

System Architecture

Figure 5 shows a schematic representation of the voxel processor architecture. Ellipses are used to represent modules running in parallel. Parallelism was achieved in several ways, the most important step is (as mentioned) to divide the object space into eight (equally sized) subspaces (FIG. 3), where each subspace has been assigned to one Transputer. The modules will be explained in more detail below.

The Controller

In the controller the user-interface software and the graphics control unit (sending commands to the Graphics subsystem) are combined. The operator communicates with the 'user-interface' part, which interprets and executes commands. The controller process is capable of sending commands to a Transputer based framegrabber, which may be used to acquire voxel data from some type of sensor. Alternatively, it is possible to read object data from subspace data, the object may be rotated and viewed interactively. The Voxel processor will produce a new image within 1 second after giving a command (for a 256*256*32 object).
Continuous rotation is possible, since the subspace transformation and the may run in parallel. The resulting images can be stored on disk, and read in again at a later time.

The Subspace Processor

This module performs the object transformation and generates a partial result for its subspace. A subspace result will be of size 256*256 for an object of 256*256*32 voxels, while the complete resulting image will be 512*512 pixels. The dimension (D) of a subspace result is easily derived from the Voxel data-base dimensions (DX, DY, DZ) :

$$D = SQRT((DX^2 + DY^2 + DZ^2))$$

A subspace partial result represents the intensity value for each pixel on the screen, except for the 'integrate' function, in which case a voxel count will be produced. The subspace data (the voxels) are loaded only once for each new object and will not be changed during the transformations. The Transputer will begin processing its data after receiving a command, which includes the transformed unit vectors and the selected type of rendering (e.g. front view, depth shade etc.). Besides performing the voxel image transformation, this module is also used for the 3D image processing operations. For this end, memory is reserved to store both the original voxel image and a filtered version.

The Merger

All partial results must be combined (merged) into the complete resulting image. This result is transferred to the controlling process where it will be stored and displayed. The 'merger' process receives eight 2D results from the subspace processors. In order to combine the partial results in a correct way, the merger needs some additional data. This data consists of :

a) A subspace offset. The offset is based on the x and y ccordinates of the subspace's transformed origin. This value is needed to place the subspace result on the correct position in the final image (FIG. 6).

b) A subspace priority. The priority is based on the z-value of the subspace's transformed origin. The lowest priority is for the subspace with the greatest distance from the viewer. The partial results of this subspace will be obscured by any subspace result of a higher priority (FIG. 7).

The merging operation is mainly performed with the 2D block-move instruction (DRAW2D) of the T800 Transputer, which can be used when the subspace results are combined into the final image in order of increasing priority (the final image is initialized to zero first). The DRAW2D can not be used for the 'integrate' function, in which case the numbers in the subspace results corresponding to the same pixel have to be added.

The Graphic subsystem

This unit is used only for controlling a framebuffer in order to display the resulting images.

Implementation Remarks

- Voxel coordinates are represented using an INT32 during transformation time. This INT32 is in fact a fixed-point real (16 bit integer part, 16 bit fraction). The accuracy is sufficient for this application and the performance is slightly better than when using REAL32.
- The object is translated to the centre of the screen, independent of the object's orientation.
- Whenever possible, special processes were assigned to communication in order to achieve maximum efficiency of the Transputer Links.
- A communication layer was integrated into the system. This layer provides data and command transport to all processes, and it is also capable of sending (debug) messages from each process to the operator screen.
- The system is very flexible in the dimensions of the objects that are to be transformed. Basically the only limitation is the memory size of each Transputer (currently 1MB). At the moment these dimensions (and a few derived values) are declared in a library. Changing this library and recompiling the software will automatically generate a new version. (N.B. The object dimensions do not have to be a power of two). Some other possible sizes which will give the same performance are : 128*128*128, 256*256*32 or 64*128*256.
- Because of the modular set-up, it is very easy to trade-off system performance against cost (a version with 4 subspace processors is also built).

Hardware

The Voxel processor is built up entirely with off-the-shelf hardware. Each processor board offers two T800's with 1MByte of memory each. Other boards used, are the Display System with a T800 and 1MByte of video-ram and finally a framegrabber with an on-board T800. Physically the system consists of a 19" cabinet with 7 single euro-sized boards installed. The host system in the Voxel processor is an IBM-AT. All the program code was written in OCCAM. For this application, it is not strictly necessary to use T800's for all processing modules (few floating-point operations are needed and only the merger uses block-moves). However, their higher link-speed does increase over-all system performance.

Current Activities

Work on the Voxel processor prototype is continued in a number of areas :

a) Addition of more 3D image-processing algorithms. An important feature will be the computer assisted image segmentation (region growing) to select interesting areas in the voxel image.
b) Implementation of 3D geometrical measurements. For medical- and biological- imaging geometrical data is very important. Surface computa tions, distances and volume measurements have to be applied to the objects in the voxel space.
c) Increase system performance by further code improvement and architecture optimization. For larger voxel space sizes a system with 16 Transputers will be developed. In this larger system the architecture will be changed to a tree structure. The advantage of a tree over a pipeline is in the shorter average length of the communication path between the PE's and the merger.
d) Implement the computation of statistical level information over (part of) the voxel data. Besides for user inspection, this has to be available to specific image processing functions like automatic thresholding, histogram equalization or edge detection.
e) Investigate (voxel) data-compression, determine effects on data transport times and implications on transform algorithms.
f) Feasibility study on stereoscopic display facilities.

Conclusions

The Voxel processor is a successful demonstration of the performance improvement and flexibility that parallel processing can deliver. It is shown that at least in some applications transputer-type parallel processing may be a good alternative for either supercomputers or dedicated hardware. Transputers have proved to be a very powerful tool. The development of the system software was not trivial but the clear representation and support of parallelism that OCCAM offers helped a lot.

The key features of the developed Voxel processor are :

- Fast, interactive system. Typical rendering speeds are 1 sec. for 2 Mbyte voxel images. The speed may be increased by using more processors.
- Highly modular and easily adaptable software. The prototype is a general purpose framework for 3D image processing.
- Low-cost, small-sized off-the-shelf hardware. Transputers are general purpose processors and the system may also be used for other (computational intensive) applications.
- Flexible performance (linear cost/performance function).

References

1) S. M. Goldwasser and R. A. Reynolds.
Real-Time Display and Manipulation of 3D Medical Objects :
The Voxel Processor Architecture. Computer Vision, Graphics and Image Processing 39.
1-27 (1987).
2) A. Kaufman and R. Bakalash.
Memory and Processing Architecture for 3-D Voxel Based Imagery.
IEEE Computer Graphics & Applications.
November 1988.
3) The Transputer Databook. INMOS publication, 1989.
4) C.A.R. Hoare (Ed). OCCAM 2 Reference Manual, Prentice Hall, 1988.
5) A. C. Tan, R. Richards, A.D. Linney.
3-D Medical Graphics - Using the T800 Transputer. Proceedings of the 8th technical meeting of the OCCAM User Group.
March 1988.

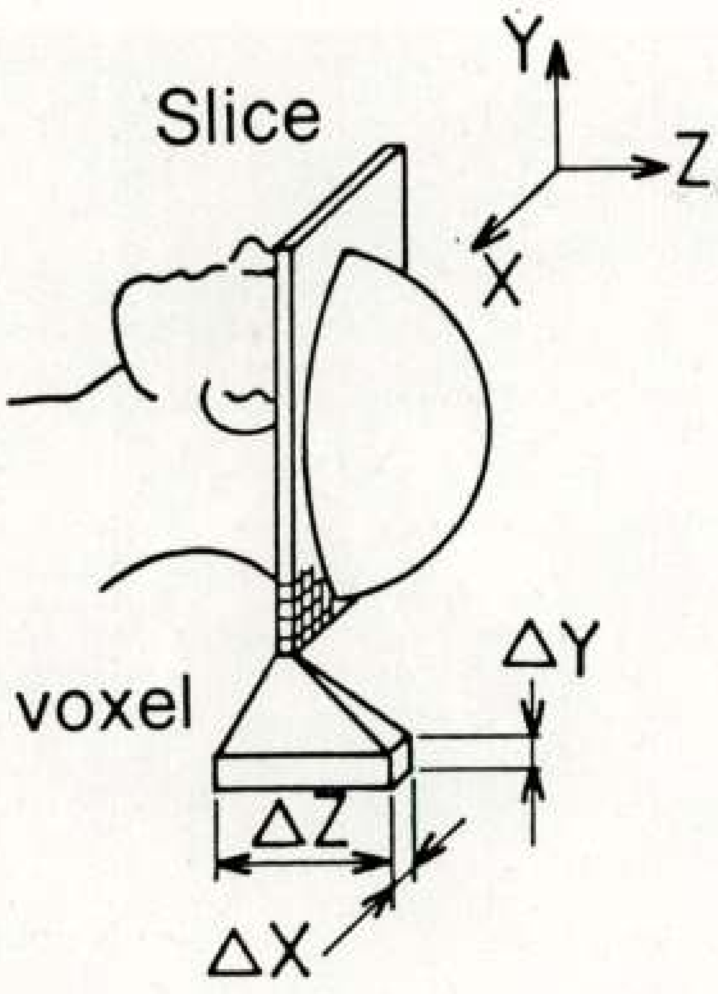


Fig. 1 Voxel Definition.

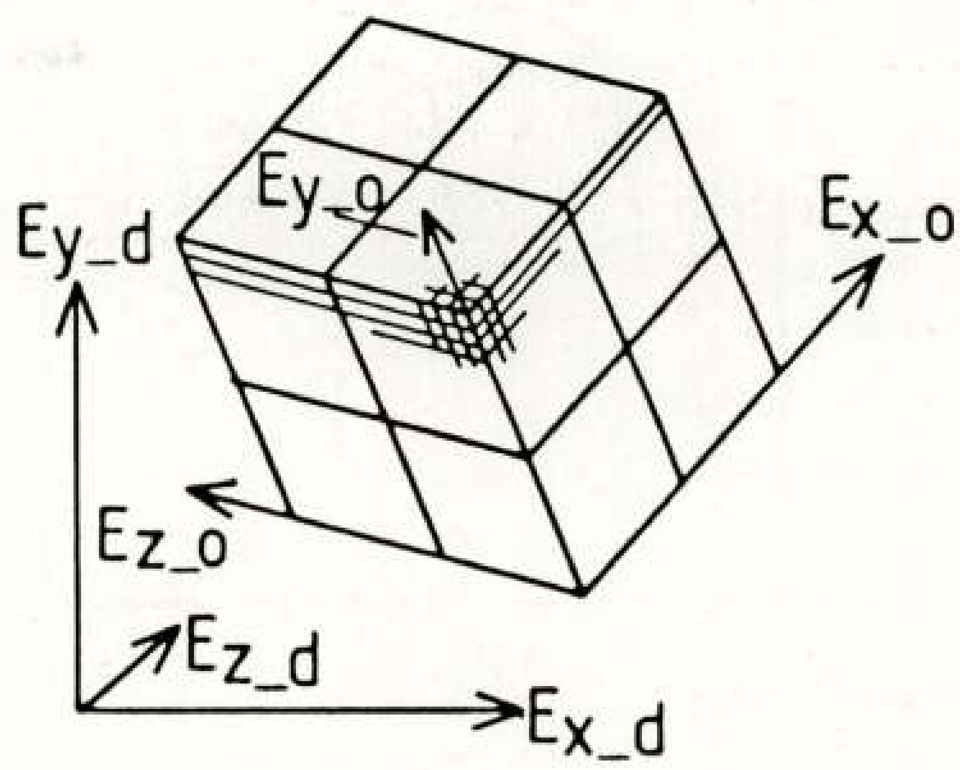


Fig. 4 Display Space.

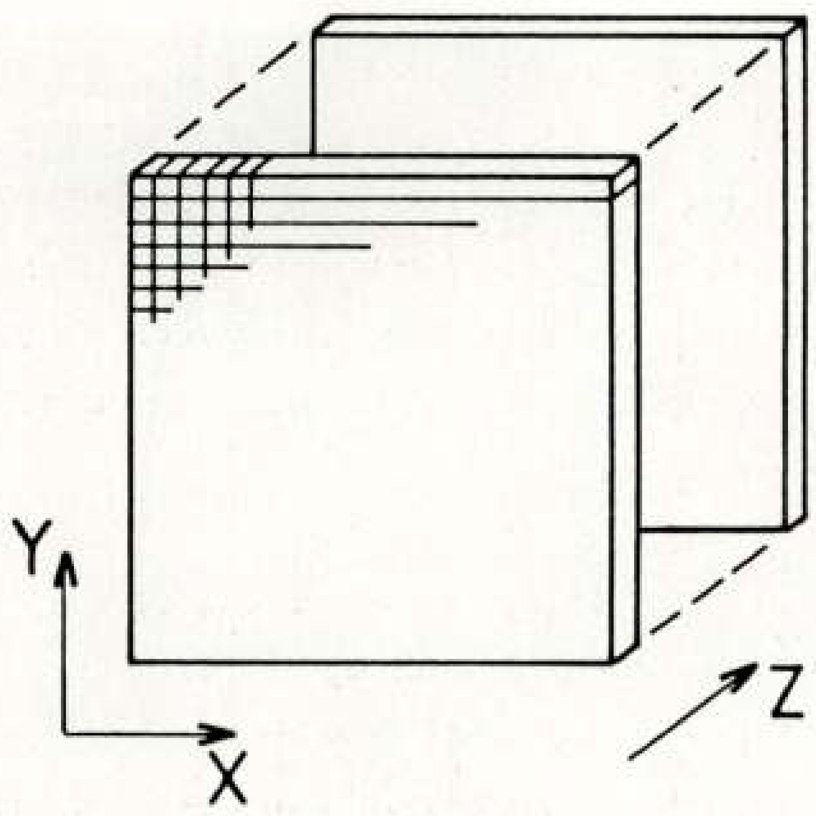


Fig. 2 Voxel Image.

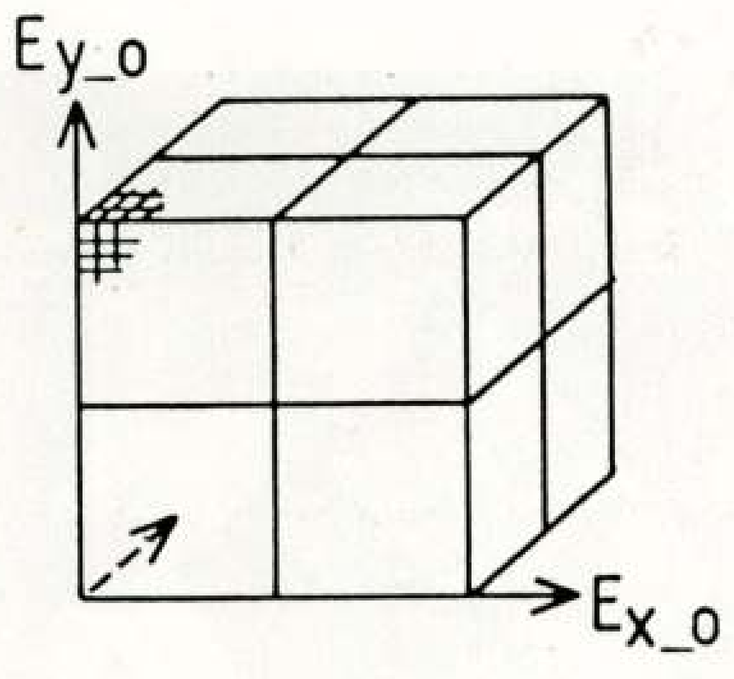


Fig. 3 Object Space.

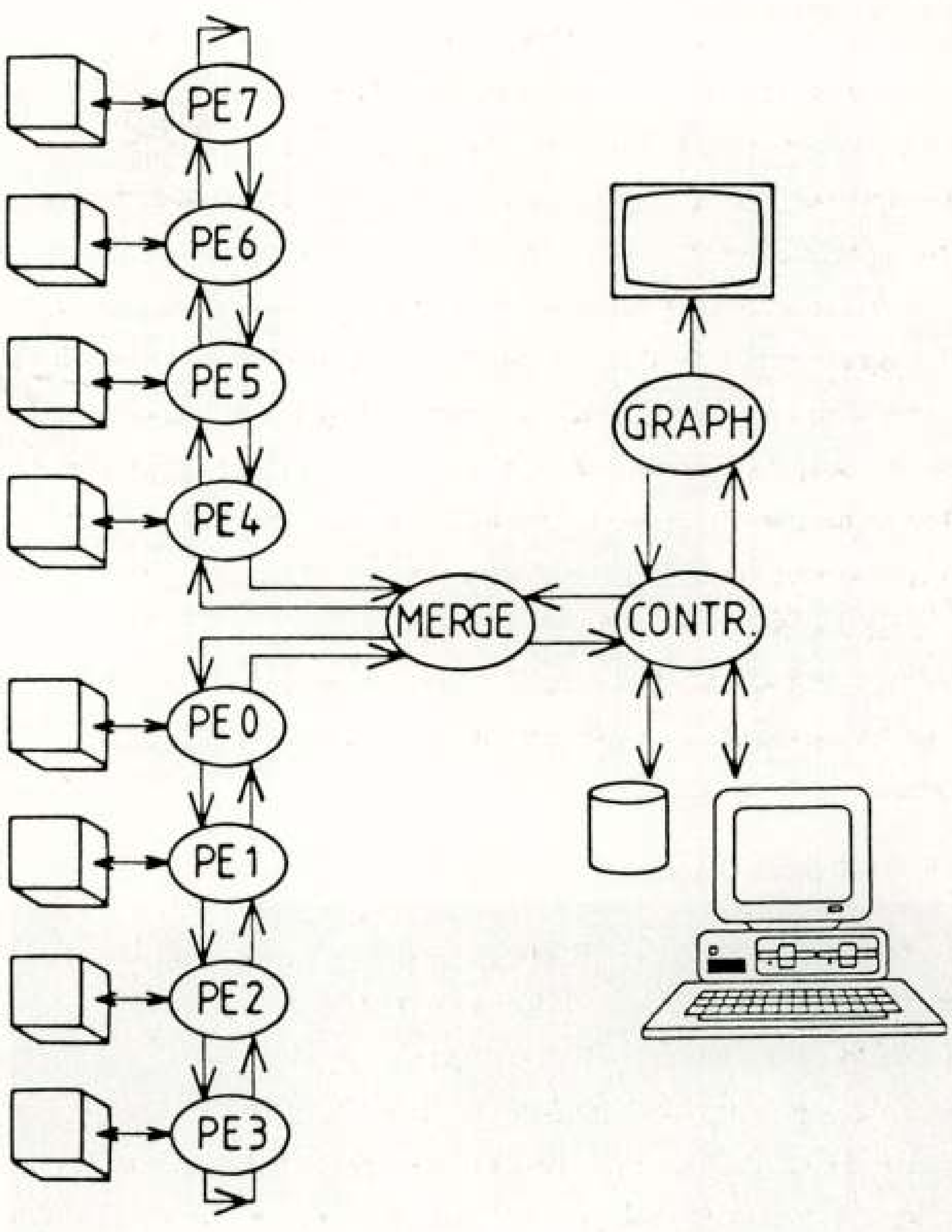


Fig. 5 System Architecture.

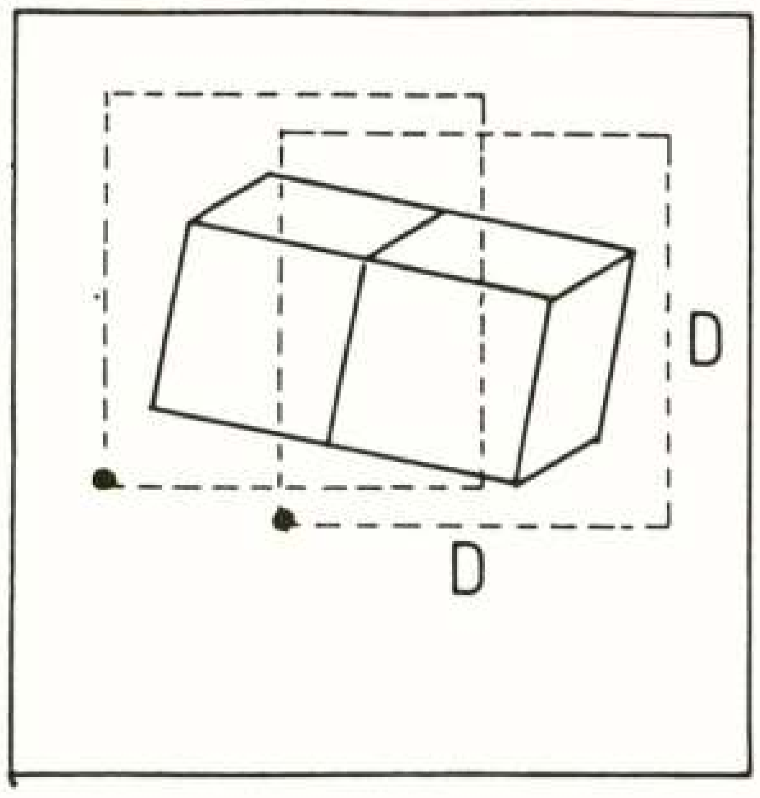


Fig. 6 Subspace Offset.

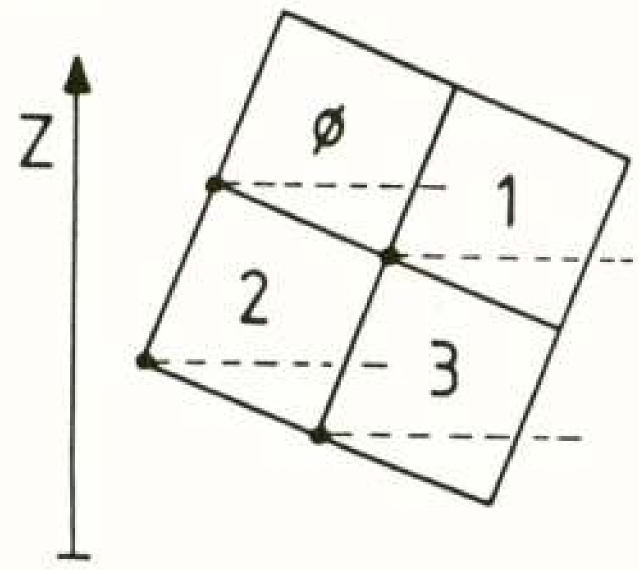


Fig. 7 Subspace Priorities.

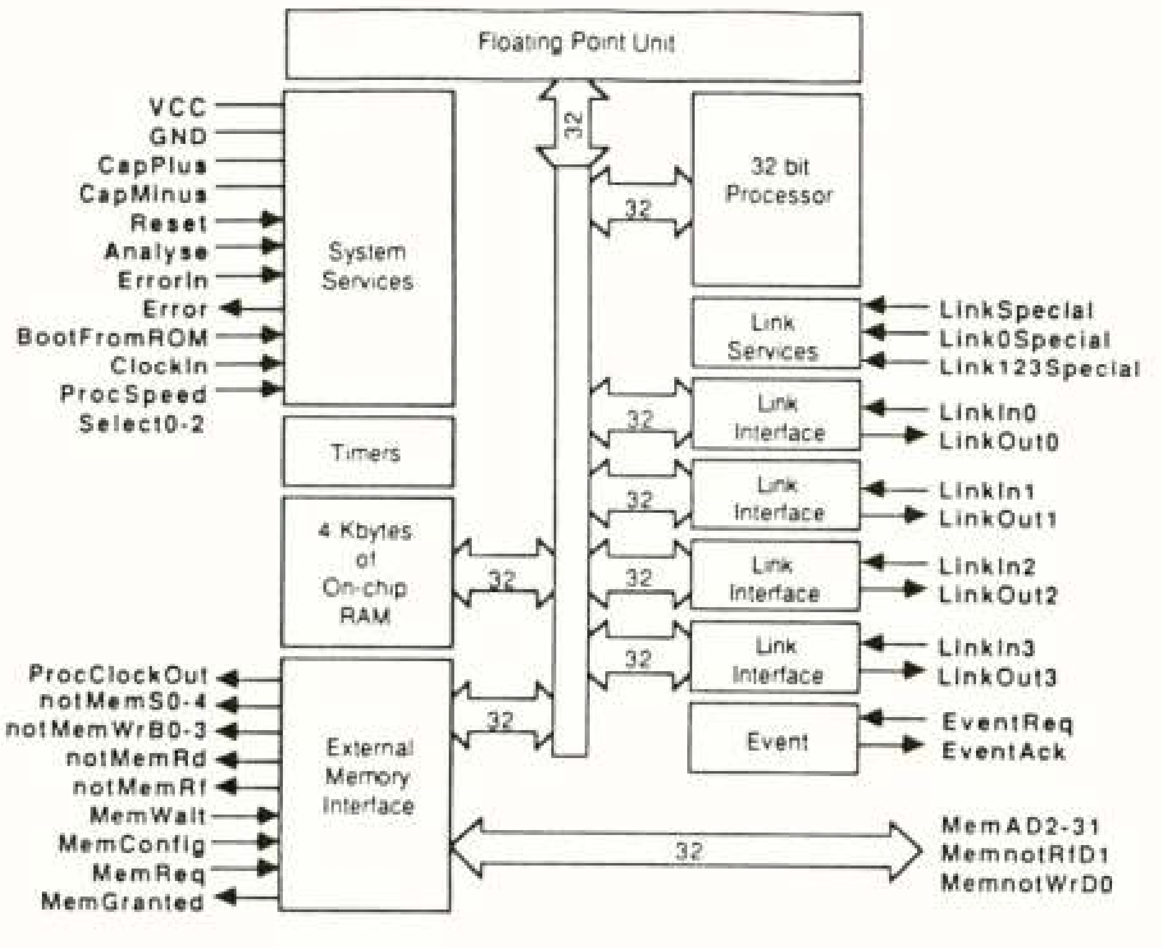


Fig. 8 T800 Block Diagram.

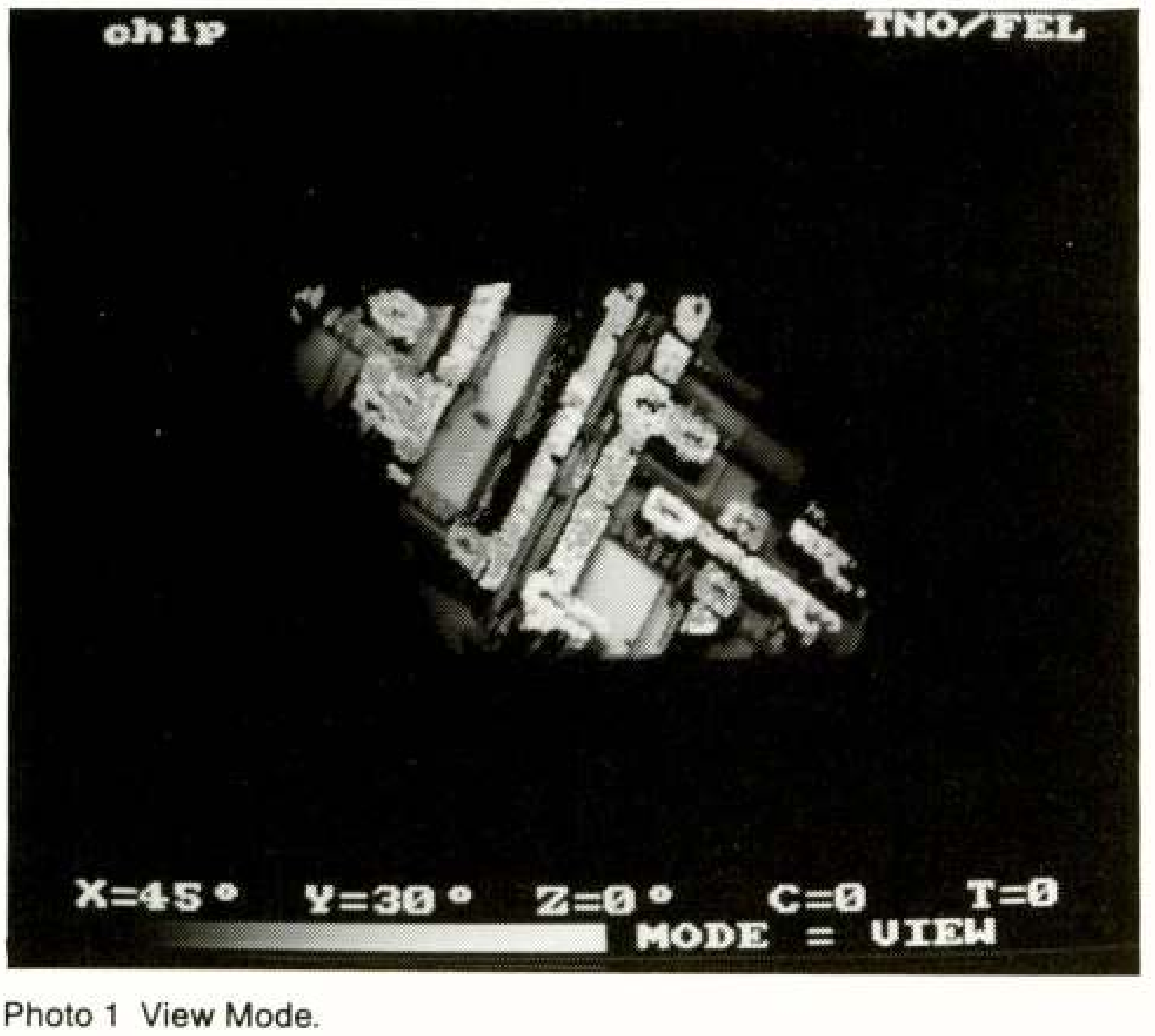


Photo 1 View Mode.

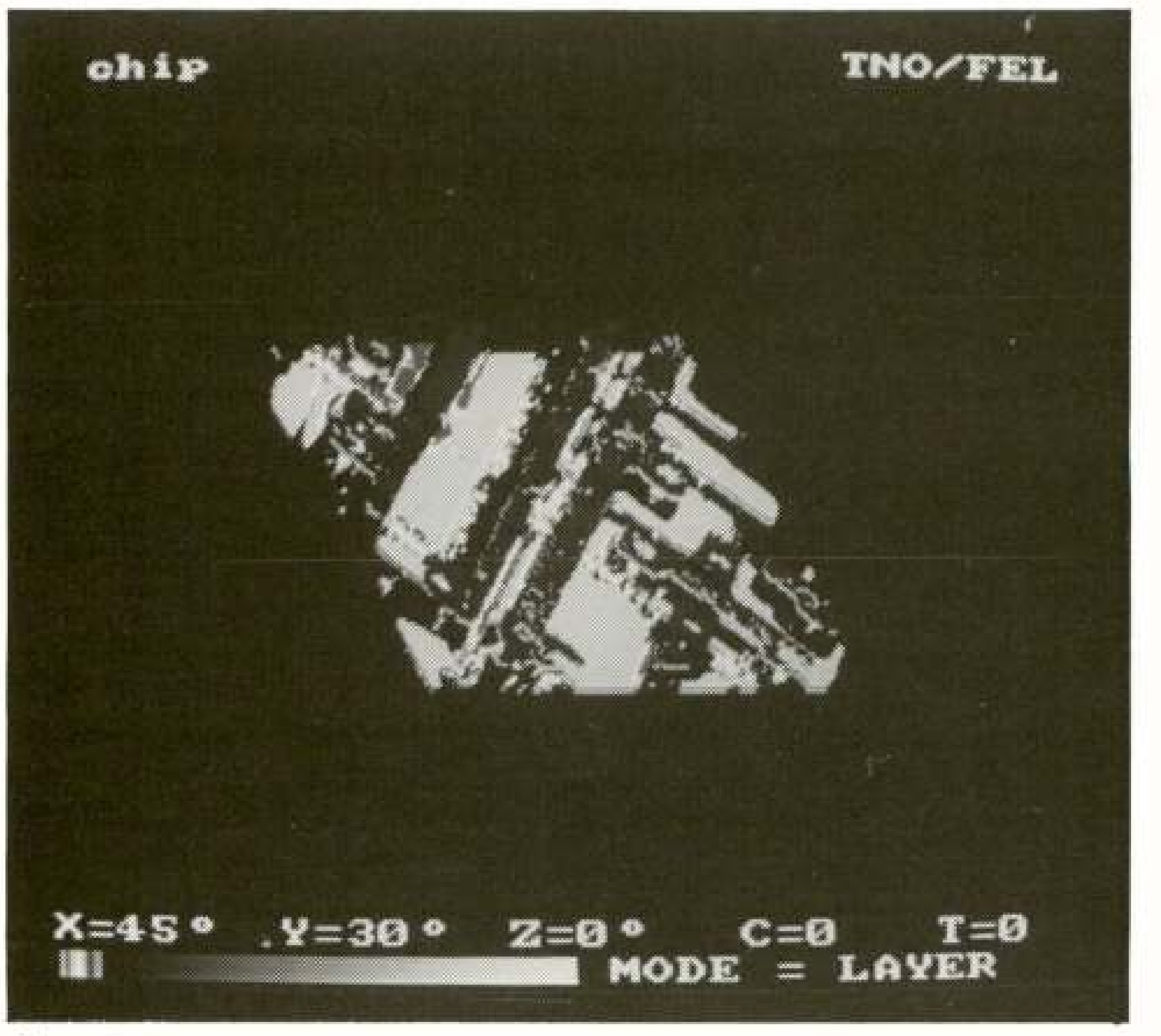


Photo 2 Layer Mode.

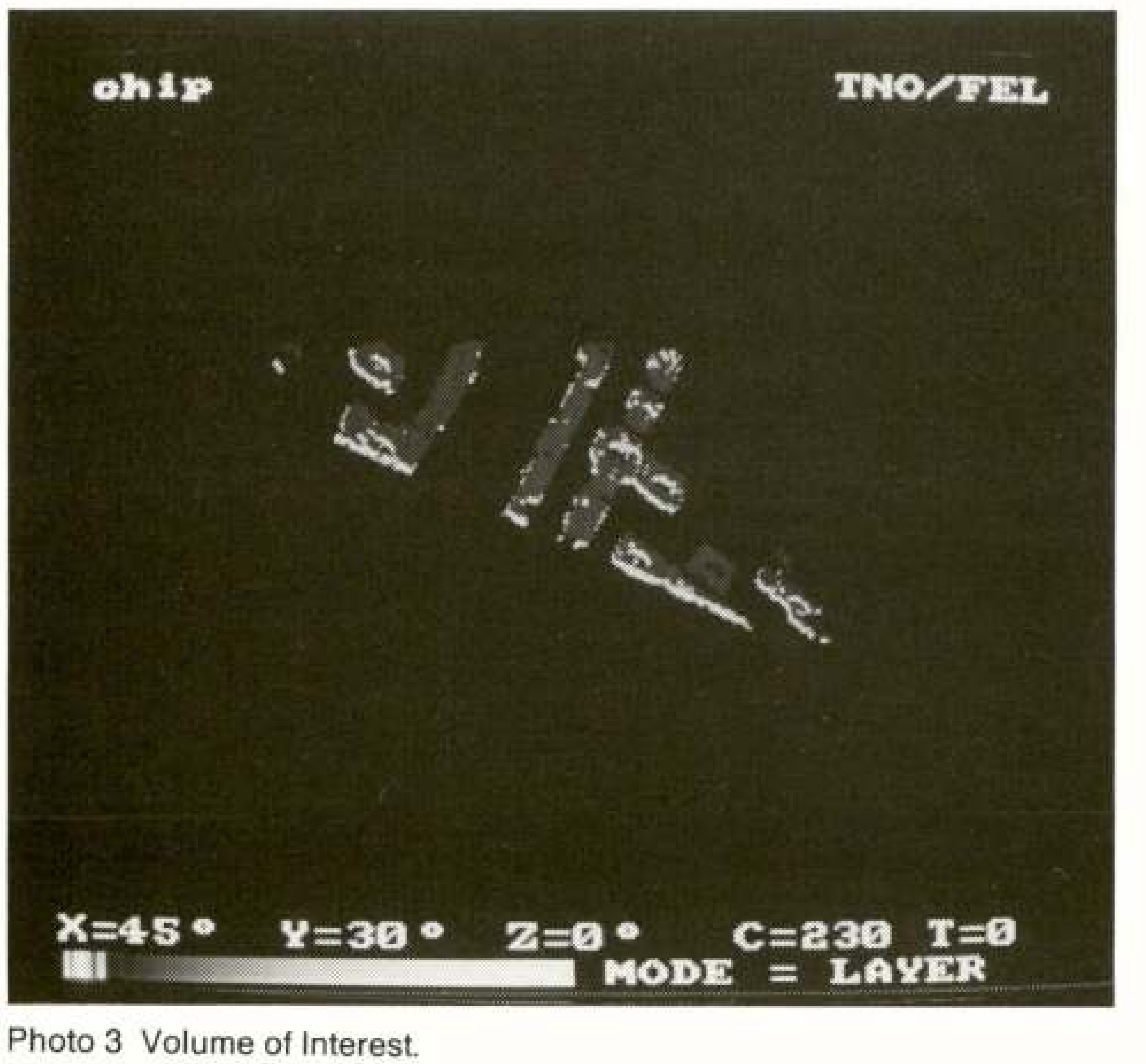


Photo 3 Volume of Interest.

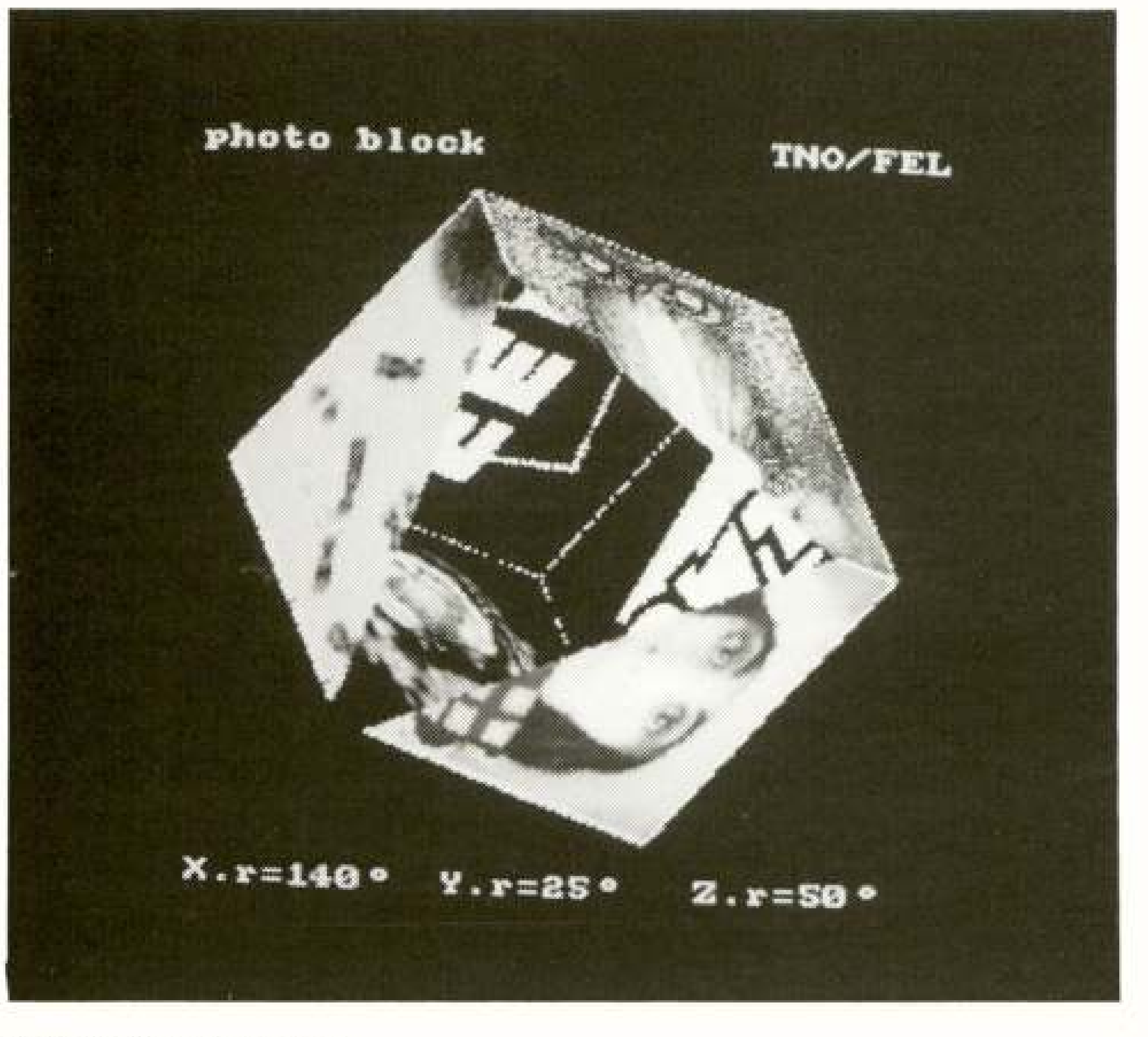


Photo 4 Cross-section.

Voordracht gehouden tijdens de 369e werkvergadering.

TRANSPUTERS, PARALLEL REKENEN IN EMBEDDED SYSTEMEN TEN BEHOEVE VAN BEELDCOMPRESSIE VOOR RUIMTEVAARTTOEPASSINGEN

Ir. M.H.J.B. Versteeg
Nationaal Lucht- en Ruimtevaartlaboratorium

Transputers, parallel processing in embedded systems for image compression used in space applications.
Within the METEOsat DISsemination (METEODIS)-project Nationaal Lucht- en Ruimtevaartlaboratorium (NLR) develops under contract with European Space Operations Centre (ESOC) and European Space Technology Centre (ESTEC) a system that demonstrates the feasibility of the use of compression and encryption within the operational METEOrological SATellite (METEOSAT)-system. The compression function is used to increase the amount of meteorological data that can be disseminated to the users.
In the METEODIS-system the compression and decompression functions are performed by programmable units. In order to perform these functions at the required speed a powerful multi processor system is required. Within the METEODIS-project these functions are realized by two times four Transputers. The Transputer is a single-chip microprocessor with a Reduced Introduction Set Computer (RISC) architecture intended for the use in multi-processor systems.

1 Introductie

Het NLR verricht al een aantal jaren onderzoek naar datacompressie-methoden voor gebruik in de lucht- en ruimtevaart. Van de recente projecten kunnen de volgende genoemd worden:

- Van 1984 t/m 1986 werd Compression and Decompression of Imaging Sensor Signals (CADISS) ontwikkeld. Dit is een prototype van een ruimtewaardig te maken multi-processor systeem voor compressie en decompressie van satellietdata.
- In 1985 is een onderzoek verricht in opdracht van ESTEC naar de mogelijkheden tot compressie van de beelddata van de METEOSAT satelliet. Het bleek mogelijk om de METEOSAT-beelddata met een factor twee te comprimeren zonder dat er bij decompressie fouten optreden ("lossless compression").

Een vervolg op dit onderzoek is het METEODIS-project. In dit project wordt in opdracht van ESTEC en ESOC een demonstratiesysteem gebouwd dat de haalbaarheid van beeldcompressie en encryptie binnen het raamwerk van het operationele METEOSAT systeem aantoont.
Voor de uitvoering van de compressie en decompressie functie worden in dit demonstratie systeem multi-processor Transputer systemen gebruikt.

2 Transputers

2.1 Multi-processor systemen

Aan computersystemen kunnen eisen worden opgelegd met betrekking tot de tijd waarbinnen de verwerkte informatie beschikbaar moet zijn. Vaak kan, ondanks de steeds sneller wordende hardware, niet aan deze eisen worden voldaan bij het toepassen van een systeem met een enkele processor. Om de verwerkingssnelheid van een systeem te vergroten kunnen meer processoren worden toegevoegd.
Een snelle communicatie tussen processoren is mogelijk wanneer alle processoren in een systeem toegang hebben tot een enkel geheugen. Deze toegang tot dat centrale geheugen vormt echter al snel een 'bottle-neck' in een systeem. Bij een aantal processoren zal namelijk de gemeenschappelijke bus verzadigd raken en het toevoegen van extra processoren heeft dan geen nuttig effect meer op de rekenprestaties. Wanneer dit punt bereikt wordt, is natuurlijk afhankelijk van het te berekenen probleem en het gebruikte algoritme.
Een oplossing hiervoor is dat elke processor toegang heeft tot een eigen geheugen terwijl een ander communicatiemedium gebruikt wordt voor het uitwisselen van gegevens tussen de processoren. Dit is de weg die in op Transputers gebaseerde systemen gevolgd wordt.
Voordat de Transputer verder besproken wordt, zal eerst de taal aan de orde komen die de basis vormt voor het model van de parallelle processen, zoals dat op de Transputer geïmplementeerd is.

2.2 De programmeertaal Occam

Software voor multi-tasking systemen kan met behulp van conventionele programmeertalen geschreven worden. De afzonderlijke processen worden dan met losse programma's beschreven, die met behulp van bijvoorbeeld communicatieprimitieven informatie kunnen uitwisselen. Deze communicatieprimitieven kunnen beschikbaar zijn in de vorm van losse routines, maar deze kunnen ook in de taal zijn opgenomen. Deze programmeertalen blijven echter in beginsel sequentiële programmeertalen (bijvoorbeeld Pascal, of C met een real-time kernel).
Meer geïntegreerde oplossingen zijn bijvoorbeeld te vinden in Concurrent Pascal en Ada, waarin speciale structuren gedefinieerd zijn voor communicatie en synchronisatie tussen de processen.

In Occam is het concept van parallelle processing

op een basisniveau ingevoerd (Pountain Ref. 3, May Ref. 4). Ieder statement in een Occam programma heeft de status van een op zich zelf staand proces. Processen kunnen worden samengevoegd tot grotere eenheden met behulp van "constructors". De belangrijkste constructors zijn de "SEQ" en de "PAR":

SEQ: De binnen de "SEQ" constructor vallende processen worden sequentieel afgewerkt, dit komt overeen met een normaal equentieel programma. De "SEQ" is pas voltooid wanneer het laatste, en daarmee alle, statements uit de "SEQ" voltooid zijn.

PAR: De binnen de "PAR" constructor vallende processen worden parallel uitgevoerd. De "PAR" is pas voltooid wanneer alle processen binnen de constructor voltooid zijn.

De toegang tot variabelen binnen een constructor is ook op het mogelijk parallel verlopen van processen afgestemd:

SEQ: Het gebruik van variabelen binnen een "SEQ" komt overeen met een sequentiële programmeertaal; alle globale variabelen mogen gelezen en beschreven worden.

PAR: Bij toegang tot variabelen binnen een "PAR" constructor moet onderscheid gemaakt worden tussen twee gevallen:
- variabelen die binnen een "PAR" door één proces beschreven worden zijn daarmee voor andere processen ontoegankelijk.
- variabelen die alleen gelezen worden zijn voor alle processen binnen een "PAR" toegankelijk.

Met deze beperking worden synchronisatieproblemen vermeden.

Voor de communicatie tussen processen bestaat binnen Occam het zogenaamde "channel": een eenrichtings-communicatiemedium. Een proces kan lezen van of schrijven naar een "channel" met behulp van speciale assignment statements. De data-transfer over een "channel" is synchroon; dat wil zeggen dat de transfer plaats vindt als beide processen gereed zijn voor de communicatieactie.

Occam sluit hierbij echter "dead-locks" niet uit. Met twee kanalen kan eenvoudig een statische "dead-lock" gecreëerd worden.

2.3 Transputer

De Transputer is een single-chip microprocessor die speciaal ontwikkeld is om het Occam parallel-processing model efficiënt te kunnen uitvoeren. De Transputer bestaat uit een 32-bits processor (de T212 bevat een 16-bits processor) met een RISC-architectuur. Qua prestaties kan de Transputer zich goed meten met andere moderne chips (Fig. 1) (1988 Ref. 1).

Om het Occam multi-tasking model te ondersteunen bevat de Transputer een scheduler die direct in microcode is vastgelegd. Hierdoor kan een proceswisseling zeer snel uitgevoerd worden (minder dan 1 μs). Het gevolg hiervan is, dat er geen keuze meer mogelijk is tussen verschillende scheduling-algoritmen.

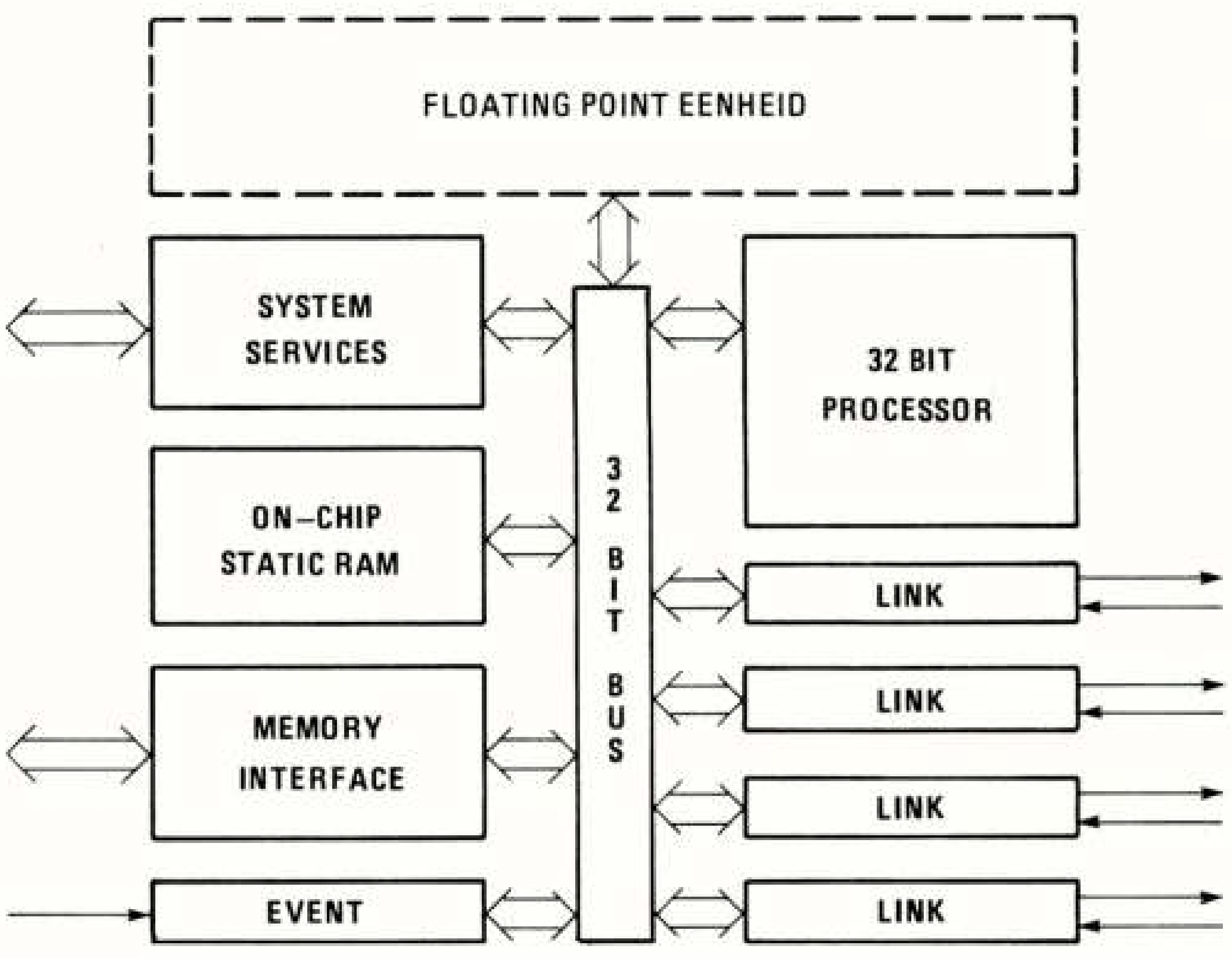


Fig. 1 Overzicht Transputer

In een Occam-programma bestaat er één beschrijvingsmethode voor "channels". De uitvoering van deze "channels" is afhankelijk van de verdeling van de processen en de links over de beschikbare Transputers:
- In het geval dat op een enkele Transputer verschillende processen parallel worden uitgevoerd zullen de "channels" tussen deze processen in het geheugen van de Transputers geplaatst zijn.
- Een Transputer bevat ook een aantal externe "links" , deze seriële verbindingen worden gebruikt als "channels" tussen processen die op verschillende Transputers worden uitgevoerd.

De toekenning van processen aan processoren in een systeem kan in een late fase van de software-ontwikkeling worden uitgevoerd, omdat de configuratie informatie los staat van het eigenlijke programma. Hierdoor is het ook mogelijk om, met een beperkte prestatie, ontwikkelde software op één of enkele Transputers te testen wanneer het benodigde multi-Transputer systeem nog niet beschikbaar is. Op dit moment zijn er nog weinig hulpmiddelen voor het verdelen van processen over de verschillende Transputers in een systeem. De meeste huidige Transputers bevatten vier seriële "links", die onafhankelijk van de processor data over de "links" kunnen verplaatsen (Direct Memory Access (DMA) transfers). De datatransfer-snelheid over de "links" bedraagt maximaal 20 Mbit/s; dit resulteert in een effectieve transfer-snelheid van maximaal 1.7 MByte/s.

Het interne geheugen van de Transputers is bij de huidige types slechts maximaal vier KByte groot. Vaak is meer geheugen vereist voor toepassingsprogramma's en data. Het geheugen van de Transputer kan worden uitgebreid met extern geheugen; de hiervoor benodigde timing signalen inclusief de eventueel benodigde refresh-sig-

nalen worden al door de Transputer zelf gegenereerd. Hierdoor blijft een enkel processing-element eenvoudig en kan bestaan uit slechts een tiental chips.
Sommige typen Transputers bevatten nog extra functies. Als voorbeeld wordt hier een floating-point-eenheid genoemd. Deze maakt het mogelijk dat de Transputer ook numerieke berekeningen met hoge snelheid uitvoert.

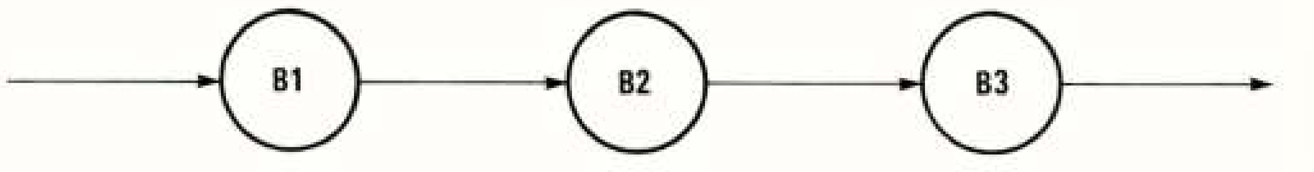
2.4 Topologie

Voor het uitvoeren van algoritmen kunnen verschillende topologiën van processen worden gebruikt. Deze procestopologiën moeten vervolgens in de uitvoeringsfase worden afgebeeld op hardware structuren.
Wanneer een algoritme in een aantal Occam-processen geïmplementeerd moet worden, bestaan er een groot aantal mogelijkheden tot opdeling. Een aantal voorbeelden wordt hier genoemd. Het eerste en het vijfde voorbeeld gaat uit van een opsplitsing in deel functies ("functional partitioning"), terwijl de andere voorbeelden de te bewerken data verdelen over een aantal processen ("data partitioning").

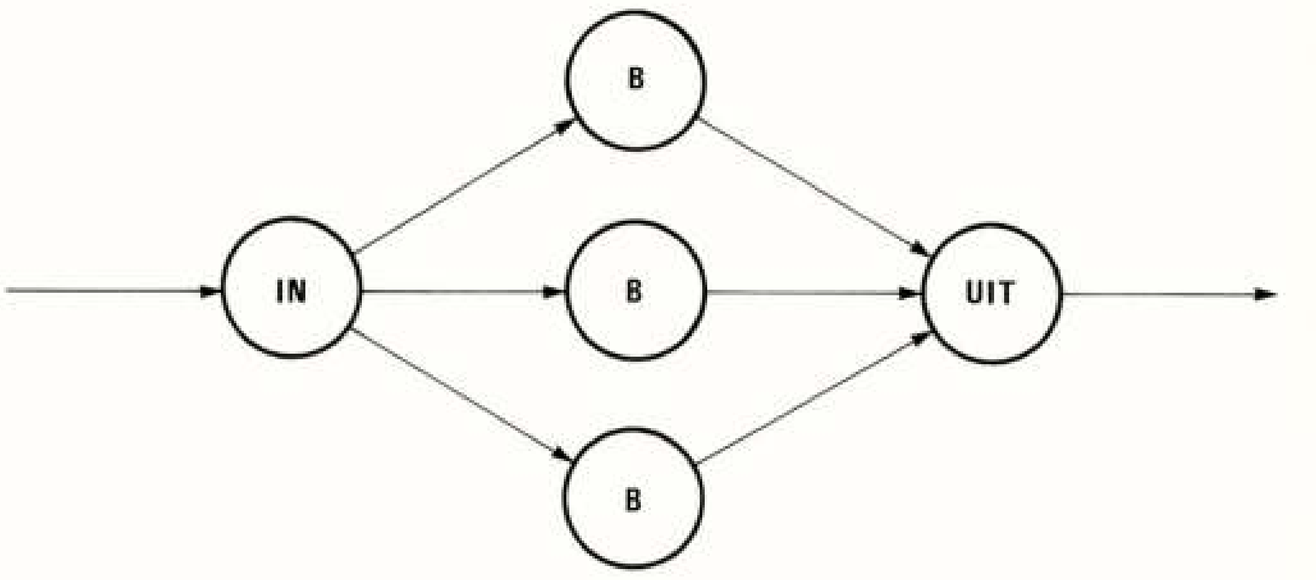
1) pipe-line:
   Een aantal processen voert ieder steeds een verschillende fase van een bewerking parallel uit. Wanneer alle processen hun data bewerkt hebben worden de datapakketten één proces doorgeschoven (Fig. 2a).
   De verschillende gekozen stappen van de totale bewerking moeten wel een vergelijkbare verwerkingstijd nodig hebben, anders zijn elementen van de pipe-line niet continu bezet.
2) parallel processing:
   Een aantal processen voert identieke bewerkingen parallel uit. Wanneer de bewerkingstijd onafhankelijk is van de ingangs-gegevens zullen de parallelle processen evenveel tijd nodig hebben voor de processtap (Fig. 2b).
   Wanneer de invoer of de uitvoer vanuit of op één punt terecht moet komen, zijn er processen nodig die de gegevens uitsplitsen of verzamelen. Gezien het geringe aantal links per Transputer kunnen maximaal drie uitgangen worden aangestuurd. Wanneer meer parallelle processing vereist is om de benodigde prestaties te bereiken, moet het uitsplitsen/ samenvoegen in verschillende trappen gebeuren.
3) processor array:
   De processoren kunnen in een één-, twee- of driedimensionaal netwerk verbonden worden, wat bijvoorbeeld overeenkomt met een logische rangschikking van de ingangs- of uitgangsgegevens. Als voorbeeld valt hier te denken aan beeldbewerking. De processoren kunnen hierbij gerangschikt worden in een twee-dimensionaal array (Fig. 2c). Dit is vooral aantrekkelijk wanneer de uit te voeren bewerkingen voornamelijk een lokaal karakter hebben.
4) farm:
   Elke eenheid voert een volledige bewerking uit maar bevat ook nog twee doorgeefprocessen (Fig. 2d).
   Deze configuratie is vooral geschikt voor bewerkingen waarvan de verwerkingstijd sterk afhankelijk is van de ingangs gegevens. Bij deze "farm"-configuratie zullen alle processoren steeds continu bezet zijn.
5) hiërarchische structuur:
   Sommige algoritmen lenen zich voor uitvoering in een hiërarchische structuur (Fig. 2e). Op laag niveau moeten de bewerkingen dan een lokaal karakter hebben terwijl op de hogere niveaus steeds meer ingangs-informatie samengenomen wordt.

Andere mogelijkheden benutten een specifieke topologie om bijvoorbeeld de procesconfiguratie aan te passen aan de ordening van de data of om een extra betrouwbaar systeem te verkrijgen:

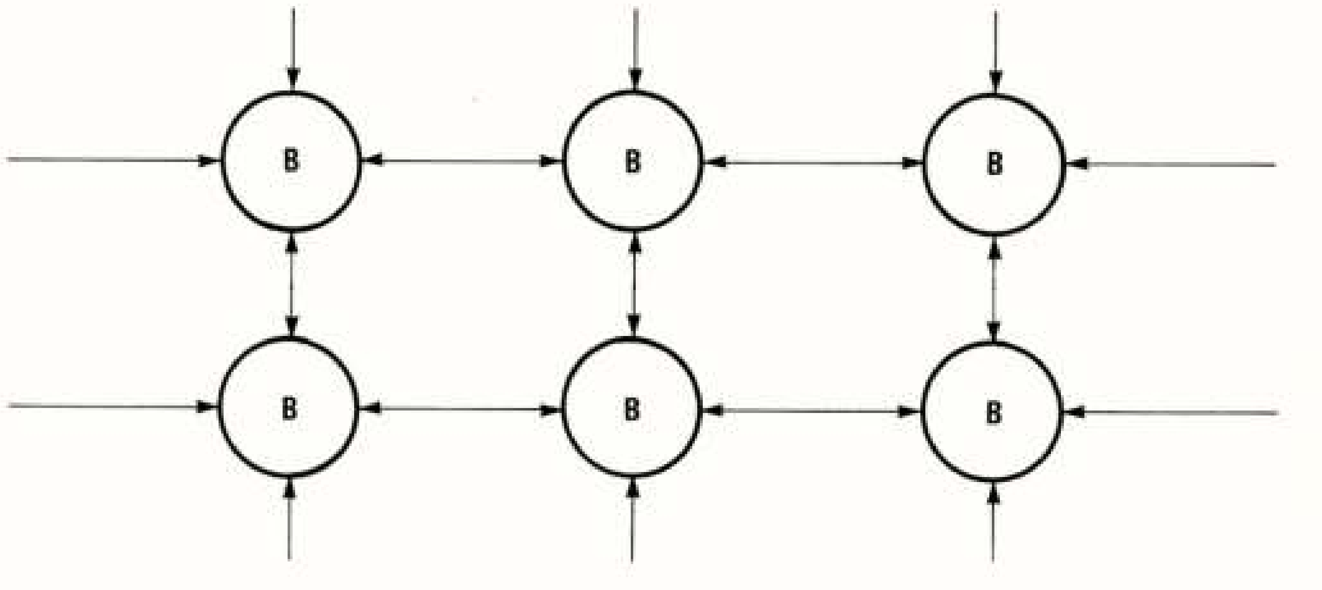
- redundant systeem:
  De topologie van een bijvoorbeeld lineair array van Transputers kan zo gekozen worden dat bij de uitval van één of enkele Transputers het gehele systeem nog steeds blijft werken (Fig. 2f). De prestaties van het gehele systeem zal dan wel terug lopen, maar de vereiste functie kan nog steeds verricht worden. Dit vereist een software-structuur die dit ondersteunt.



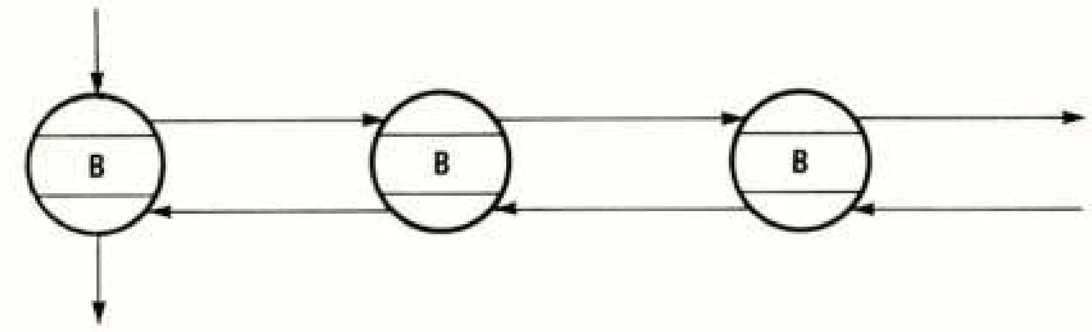
a) Pipe-line bewerking



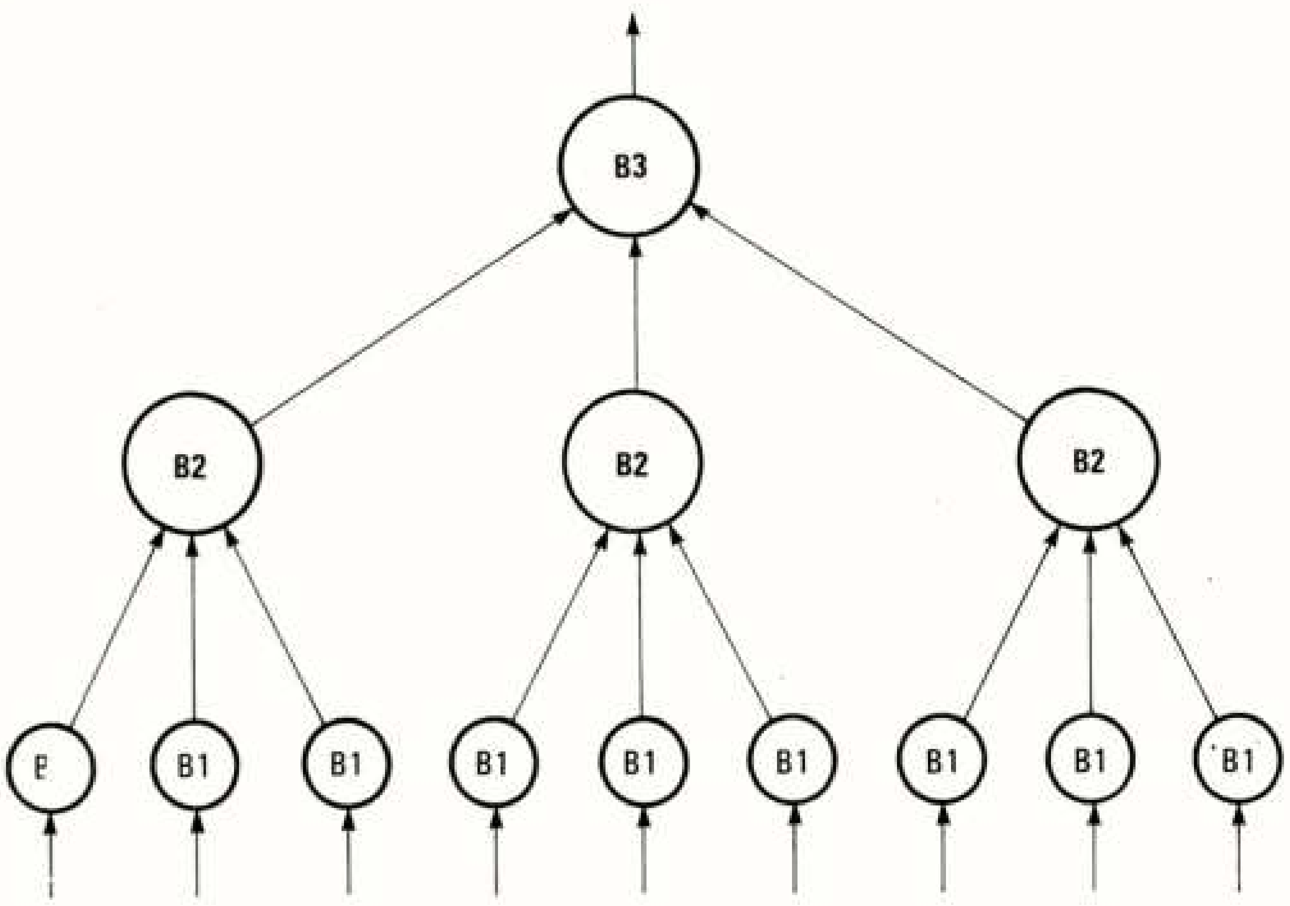
b) Parallel bewerking



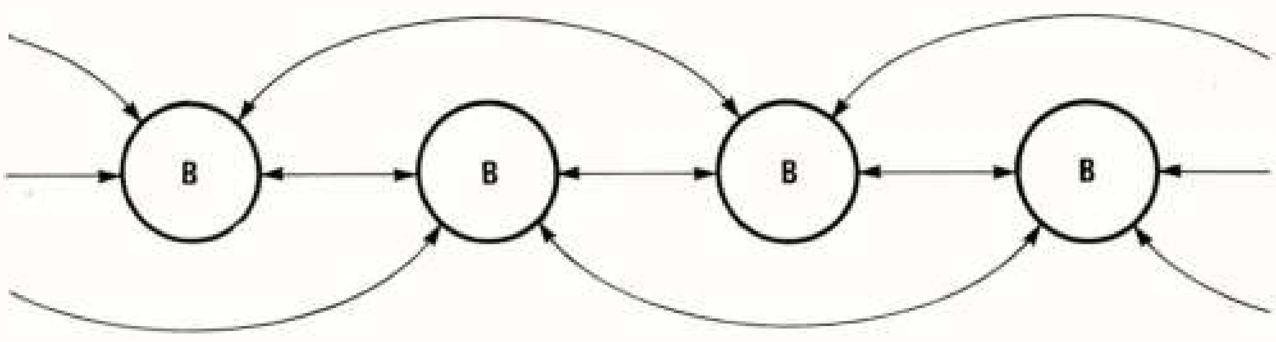
c) Twee- dimensionaal netwerk



d) 'Farm' bewerking



e) Hierarchische bewerking



f) Redundant lineair netwerk

Fig. 2 Proces-topologiën

Binnen het Transputer multi-tasking systeem kan voor de processen zonder beperking een topologie gekozen worden. Voor de hardware zijn echter beperkingen aanwezig, omdat iedere Transputer maar een beperkt aantal "links" bezit. Er zijn dan twee oplossingen:

- Verschillende logische links kunnen via een enkele fysieke link gemultiplexed worden.
- Een enkele logische link kan over een aantal fysieke links gerealiseerd worden.

Deze oplossingen gaan ten koste van de bereikbare datatransfer snelheid. Voor zo hoog mogelijke prestaties wordt er vaak naar gestreefd om het netwerk van processen zo op de hardware of te beelden dat één logische link overeenkomt met één fysieke "link".

3 METEODIS systeem

3.1 METEOSAT

De METEOSAT satelliet is een meteorologische geostationaire satelliet die boven de nul meridiaan staat. In drie spectrale banden (visueel, waterdamp en infrarood) maakt deze satelliet ieder half uur een opname van de gehele zichtbare aardbol. Afhankelijk van de spectrale band heeft dit beeld een resolutie van 5000*5000 of 2500*2500 beeldpunten.
Ieder half uur wordt zo 37½ MByte beeldinformatie verzameld. Continu wordt deze beeldinformatie naar het grondstation van ESOC in Darmstadt gestuurd (Fig. 3).

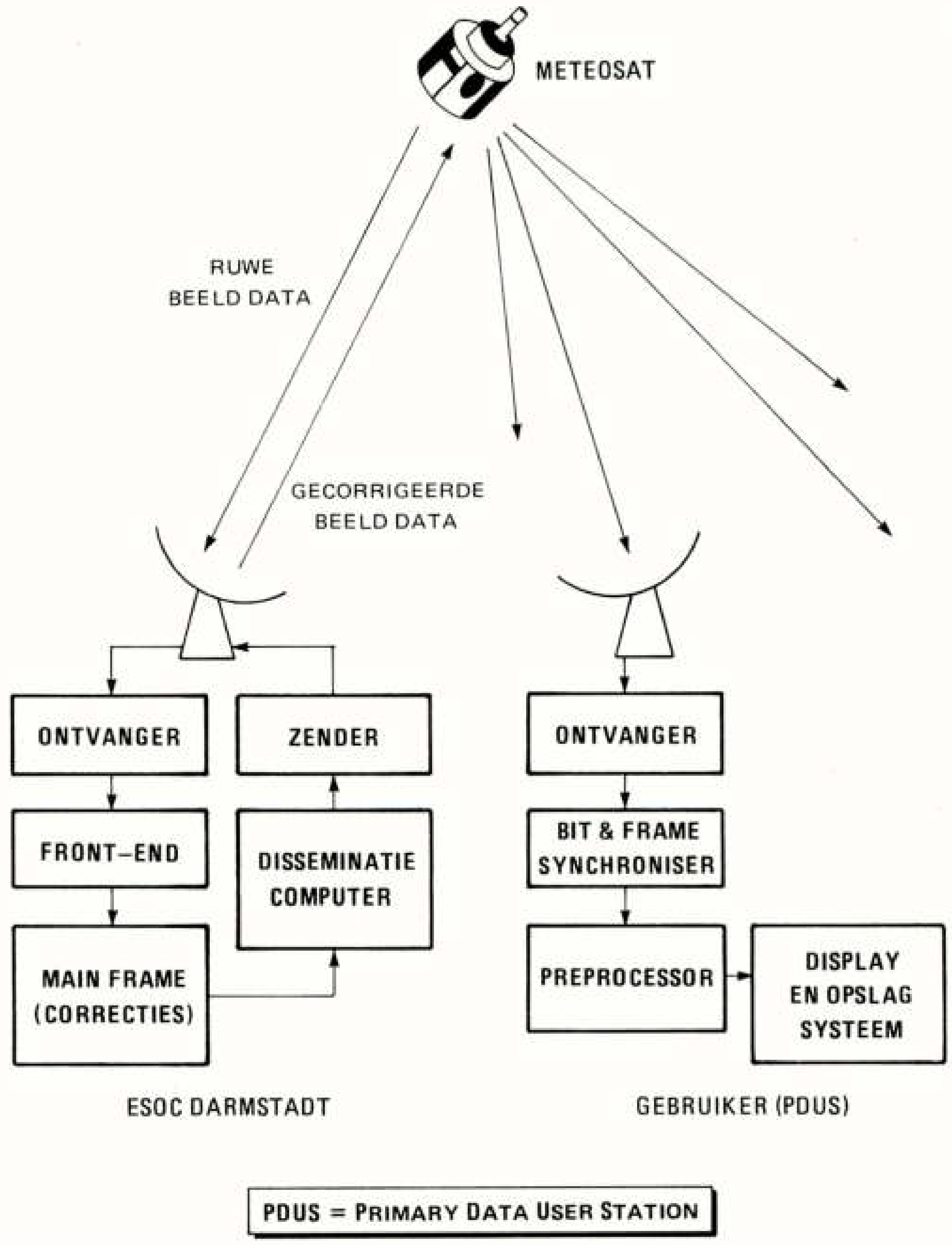


Fig. 3 Overzicht METEOSAT systeem

Hier worden op een main-frame geometrische en radiometrische correcties op de beelddata uitgevoerd. Een selectie van de beelddata wordt vervolgens via de METEOSAT-satelliet, die dan als communicatiesatelliet fungeert, naar de gebruikers verzonden.
Niet alle verzamelde beeldinformatie kan naar de gebruikers verzonden worden, omdat de capaciteit van het disseminatie-kanaal beperkt is tot 166.6 Kbit/s. Een deel van de capaciteit wordt ook nog gebruikt voor het versturen van andere produkten naar de gebruikers, zoals beelden die door de Amerikaanse Geostationary Operational Environmental Satellite (GOES) satelliet verzameld zijn en van de METEOSAT-data afgeleide produkten.

3.2 METEODIS

In het verleden heeft het NLR onderzocht hoe de METEOSAT beelddata kunnen worden gecomprimereerd. Van gebruikerszijde is hierbij altijd geëist dat compressie alleen interessant is, als er na reconstructie geen

fouten overblijven ten opzichte van het oorspronkelijke beeld. Een mogelijkheid tot een gemiddelde compressiefactor van ruim twee werd gevonden in het MEANDER-compressie-algoritme. De uitgevoerde studie betrof vooral de theoretische mogelijkheden tot datacompressie.
Het METEODIS-project zal de haalbaarheid van compressie binnen het operationele METEOSAT-systeem aantonen (Börger 1988). Het METEODIS-systeem is een demonstratiesysteem, dat mogelijk kan uitgroeien tot een compleet operationeel systeem. Naast de compressiefunctie zal ook een encryptiefunctie gerealiseerd worden, maar deze zal hier verder niet besproken worden.
De compressie- en decompressie-functies van het METEODIS-systeem worden gerealiseerd in twee embedded systemen (Fig. 4):

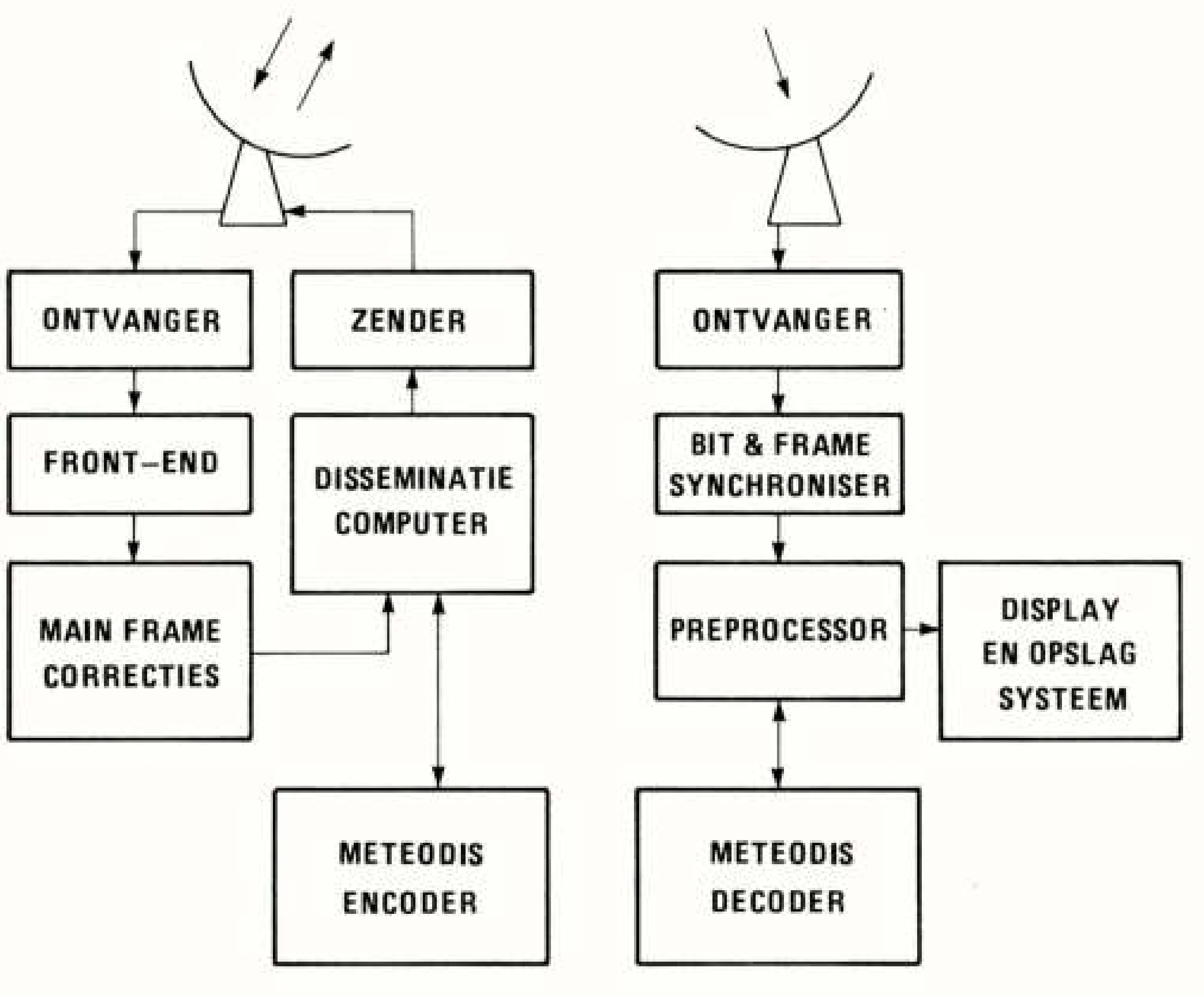


Fig. 4 METEODIS demonstratie systeem

- Het eerste systeem met de compressiefunctie, wordt verbonden met de computers in Darmstadt en zal de beelddata comprimeren voordat een beeld wordt verstuurd naar de gebruikers.
- Het tweede systeem vormt een prototype van het embedded-systeem dat iedere gebruiker met zijn operationele ontvangstsysteem moet verbinden. Voor de demonstratie wordt een ontvangstsysteem in Darmstadt met deze eenheid uitgerust.

Beide systemen bevatten geen andere interfaces dan de bovengenoemde. De data die gecomprimeerd moeten worden zullen een indicatie bevatten die aangeeft dat deze functie geactiveerd moet worden. Deze indicator wordt ook meeverzonden met de data over de satellietverbinding, zodat het decodeersysteem aan de decompressiefunctie kan starten.
Voor de gebruiker van de METEOSAT-beelden is dit gehele systeem transparant. Het enige dat hij/zij merkt is dat de beelddata in kortere tijd verzonden kan worden.

Na compressie is een groot deel van de redundante informatie uit de beelddata verwijderd. Hierdoor zijn deze data gevoeliger voor transmissiekanaalfouten. Als kanaalfouten de verzonden data aantasten, zullen de effecten op de beelddata na decompressie ook veel beter zichtbaar zijn: een enkel verminkt bit kan nu een deel van een beeldlijn verstoren, terwijl zonder compressie slechts een enkel beeldpunt zou zijn aangetast.
Om ervoor te zorgen dat de kwaliteit van de beelddata die de gebruiker bereikt niet achteruitgaat, wordt een Forward Error Correcting (FEC) code toegepast.
Bij metingen van het transmissiekanaal bleek dat de gevonden kanaalfouten konden worden geclassificeerd als random single bit errors. Daaruit volgde dat met een eenvoudige FEC-code volstaan kon worden. Als er bijvoorbeeld sprake was geweest van burst errors, bit insertions en/of bit deletions, was een complexere FEC nodig geweest. Deze code voegt gecontroleerd enige redundantie (≈ 5 %) aan de verzonden data toe, waardoor aan de ontvangende kant een deel van de opgetreden kanaalfouten hersteld kan worden.

4 Algoritmen

4.1 Compressie

Voor de compressie van de METEOSAT-beelddata wordt het MEANDER-algoritme gebruikt. Met dit algoritme kunnen beelddata gecomprimeerd worden en vervolgens weer foutloos gedecomprimeerd. Foutloos wil hier zeggen dat er dan geen verschil is tussen het gedecomprimeerde beeld en het origineel. Voor deze klasse van compressie-algoritmen ("lossless") is de behaalde compressiefactor echter afhankelijk van de informatie-inhoud van de data (Fig. 5). Aangezien deze over het beeld varieert, kan het best worden gesproken over de gemiddelde compressiefactor. In het geval van de METEOSAT-beelden kan voor alle drie spectrale kanalen een gemiddelde compressiefactor van ruim twee gehaald worden.

Het algoritme werkt op stukken van beeldlijnen en bestaat uit de volgende stappen:

Verschillen bepalen (DPCM):

Van opeenvolgende beeldpunten worden de verschillen bepaald. Omdat er sprake is van correlatie tussen opeenvolgende beeldpunten, zullen deze verschillen voornamelijk uit kleine getallen bestaan. De waarde van het eerste beeldpunt van een lijnstuk wordt direct verzonden.

Indelen in klassen:

De verschillen worden in klassen ingedeeld, die aangeven hoeveel bits benodigd zijn om de verschillen te coderen. De klassen worden zo gekozen dat het verschil tussen opeenvolgende klassen maximaal één is.

Klassen coderen:

De verschillen tussen opeenvolgende klassen worden per drietal gecodeerd en verzonden; de eerste klasse wordt weer direct verzonden. Voor het coderen van deze drietallen wordt een Huffman-code-

ring gebruikt, de hiervoor gebruikte coderingstabel is afhankelijk van de soort (spectrale kanaal) beelden.

Verschillen coderen:

De verschillen worden gecodeerd met het aantal in de klasse gespecificeerde bits. Een kleine winst word nog geboekt door verschillen waarvan bekend is dat deze optimaal gecodeerd kunnen worden, met een bit minder te coderen.

Het eerste beeldpunt, de gecodeerde klassen en de gecodeerde verschillen worden vervolgens samengevoegd en verstuurd.

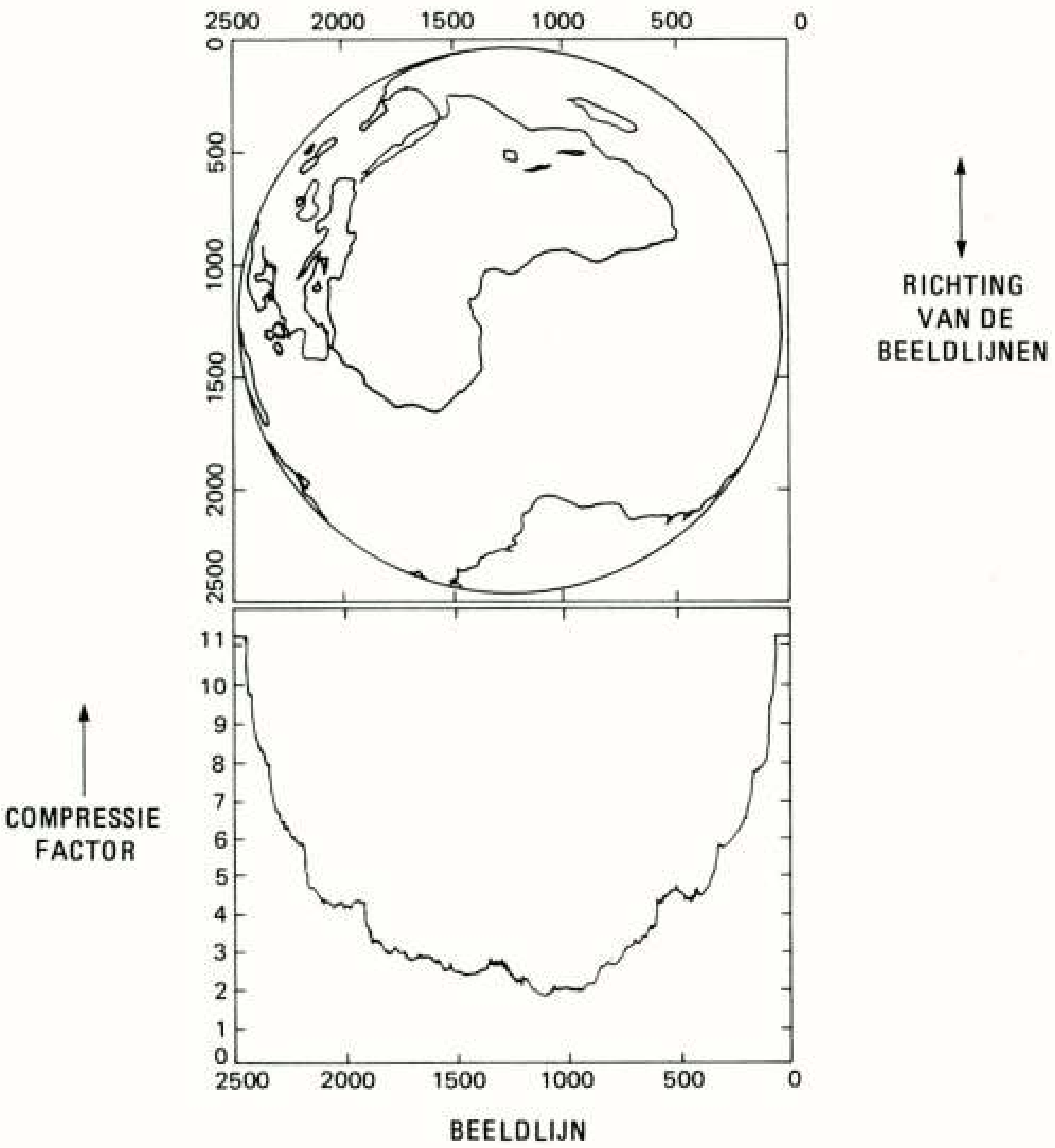


Fig. 5 Compressie factor afhankelijk van de beeld informatie

Het compressie-algoritme codeert stukken beeldlijn met een vaste lengte. Hierdoor ontstaan gecomprimeerde segmenten met een variabele lengte. Het METEOSAT-transmissieformaat gaat echter uit van frames met een vaste lengte. De segmenten met een variabele lengte worden, tezamen met een identificatie, in de frames verpakt (Fig. 6). In elk frame wordt nog een aantal ruimtes vrijgehouden voor de nog toe te voegen FEC code.

Om ook na korte onderbrekingen van een uitzending het begin van gecomprimeerde segmenten te kunnen vinden wordt er per frame een pointer naar de start van het eerst volgende gecomprimeerde segment toegevoegd. Deze pointer verzorgt de synchronisatie van de verzonden segmenten. De frames met een vaste lengte bevatten een hardware synchronisatiepatroon dat door de ontvanger gedetecteerd wordt en ervoor zorgt dat het begin van de frame in een transmissie gevonden kan worden.

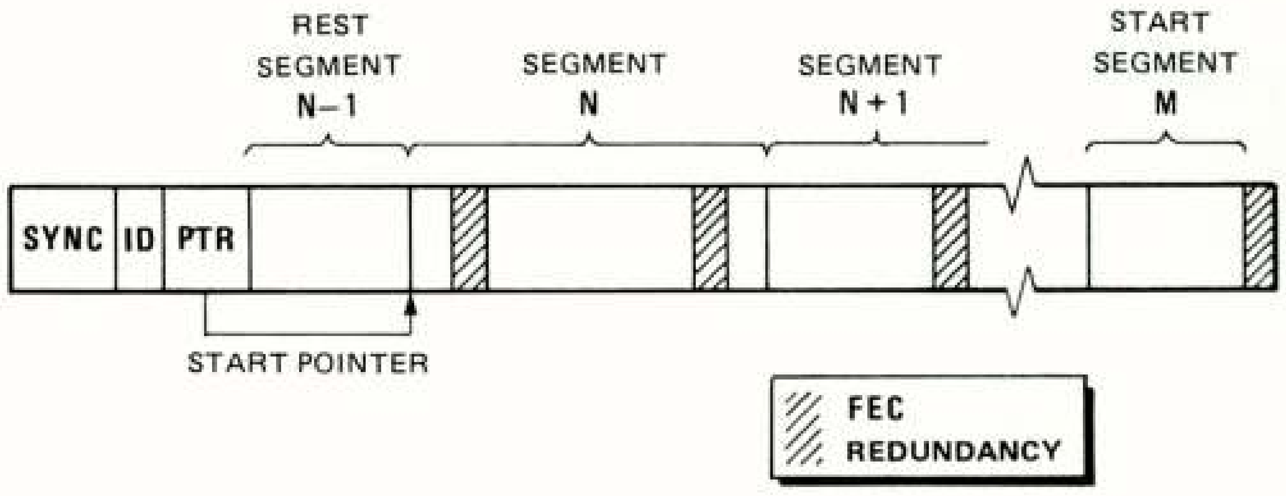


Fig. 6 Verpakken van variabele lengte segmenten in vaste lengte frames

4.2 Foutcorrectie (FEC)

Om bij de ontvanger foutcorrectie mogelijk te maken wordt aan de te verzenden data een Forward Error Correction (FEC) code toegevoegd. Aan de ontvangende zijde kan nu een deel van de opgetreden kanaalfouten gecorrigeerd worden.

De FEC code werkt op blokken data van een vaste lengte (171 bits), aan elk blok wordt door de FEC-code een aantal redundante bits toegevoegd. Deze bits zijn een soort 'pariteits'-bits, die bepaald worden uit de deling van de te versturen data met het FEC-generatorpolynoom.

Aan de ontvangende zijde wordt een overeenkomstige bewerking uitgevoerd. Er kan dan gedetecteerd worden of de ontvangen data correct (foutloos) is ontvangen. Als er tijdens het versturen van het blok één enkele transmissiefout is opgetreden kan deze met behulp van de redundante bits hersteld worden. De gekozen FEC biedt ook nog de mogelijkheid om te detecteren als twee bits in een verzonden blok foutief overgekomen zijn.

5 Uitvoering

Voor de uitvoering van de compressiefunctie binnen het METEODIS systeem is gekozen voor een oplossing gebaseerd op Transputers, waarbij de FEC functie in een Programmeerbaar Logic Device (PLD) is uitgevoerd.

De volgende overwegingen hebben hierbij een rol gespeeld:

- Voor het compressie-algoritme ging de voorkeur uit naar een programmeerbaar systeem, zodat kleine aanpassingen in het algoritme nog aangebracht konden worden. Ook is het gehele compressie-algoritme zo complex dat een directe hardware uitvoering erg omvangrijk zou worden.
- De gebruikte componenten moesten voldoende performance leveren om de bewerking bestaande uit de compressie en het FEC algoritme real-time (166.6 Kbit/s) uit te voeren.

  Zowel een multi-processor systeem gebaseerd op de 68000-familie (Versteeg 1987), als een multi-processor Transputer systeem kan deze taak uitvoeren. De processor eenheden bij een Transputer multiprocessor systeem bestaan uit slechts een tiental componenten zodat het gehele systeem compact kan worden uitgevoerd.

- Het moet eventueel mogelijk zijn om de compressiefunctie later aan boord van een satelliet uit te voeren. De gebruikte componenten zouden dit niet bij voorbaat onmogelijk moeten maken. In opdracht van ESTEC is een onderzoek verricht naar de mogelijkheid om ruimtewaardige Transputers te verkrijgen. Een resultaat van dit onderzoek was dat deze kwalificatie mogelijk was en dat de in de Transputers gebruikte CMOS-techniek direct al goed bestand is tegen straling.
- De deling die voor de FEC benodigd is, kan eenvoudig worden uitgevoerd met behulp van een schuifregister met terugkoppelingen. Nadat het gehele te beschermen data- woord door dit schuifregister geschoven is, bevat het schuifregister de gewenste 'pariteits'-bits; deze worden dan aan de te verzenden data toegevoegd.

  De schuifregisters met terugkoppelingen kunnen alleen met veel moeite in een processor worden uitgevoerd. Een hardware-oplossing voor de FEC, bestuurd vanuit de software, vereist slechts schuifregisters van negen elementen met een aantal terugkoppelingen.
- Het NLR had enige ervaring met Transputers; op grond van experimenten werd verondersteld dat de compressiefunctie met vier Transputers te realiseren moest zijn. Mocht echter gedurende het METEODIS-project blijken dat dit aantal niet toereikend is om de vereiste prestaties te halen, dan kan dit aantal eenvoudig worden uitgebreid.

De verdeling van de processen over de Transputers is niet eenvoudig vanwege de volgende feiten:

- Een "functional partitioning" van het algoritme is moeilijk: de beschreven algoritmestappen kunnen in een aantal gevallen nog wel wat verder opgesplitst worden, maar het aantal Occam-processen blijft toch zeer beperkt.
- Een "data-partitioning" levert een ander probleem: omdat de hele compressie functie één uitgaande data stroom moet produceren, moet er ook sprake zijn van een enkel 'pack'-proces.

In het METEODIS project is gekozen voor een "pipe-line" benadering (functional partitioning) voor de verdeling van de processen over de Transputers (Fig. 7), De Transputers voeren na elkaar de bewerkingen uit op de data. Als de keten uit veel deelprocessen bestaat kan een vergroting van de bewerkings snelheid eenvoudig bereikt worden door meer Transputers te gebruiken. Het langzaamste proces in de keten bepaalt nu de maximaal te behalen snelheid van de gehele keten. Om dan een zo hoog mogelijke verwerkingssnelheid te halen moet het langzaamste proces een eigen Transputer toegewezen krijgen.

Nadat het langzaamste proces een eigen Transputer toegewezen gekregen heeft, kan een verdere versnelling nog op twee wijzen bereikt worden:

- Het langzaamste proces wordt opgesplitst in een aantal deelprocessen die verdeeld worden over verschillende Transputers.
- Twee Transputers worden ingezet om het langzaamste proces parallel te gaan uitvoeren.

Het is bij het gebruikte algoritme niet eenvoudig mogelijk om de gehele bewerking in twee parallelle takken uit te voeren. De tijd-rovende 'pack'-bewerking die de gecomprimeerde segmenten met een variabele lengte in de transmissieframes verpakt moet door een enkel proces aan het eind van de keten worden uitgevoerd. Voor het huidige systeem wordt echter verwacht dat de gebruikte vier Transputers voldoende zijn om de vereiste snelheid te bereiken.

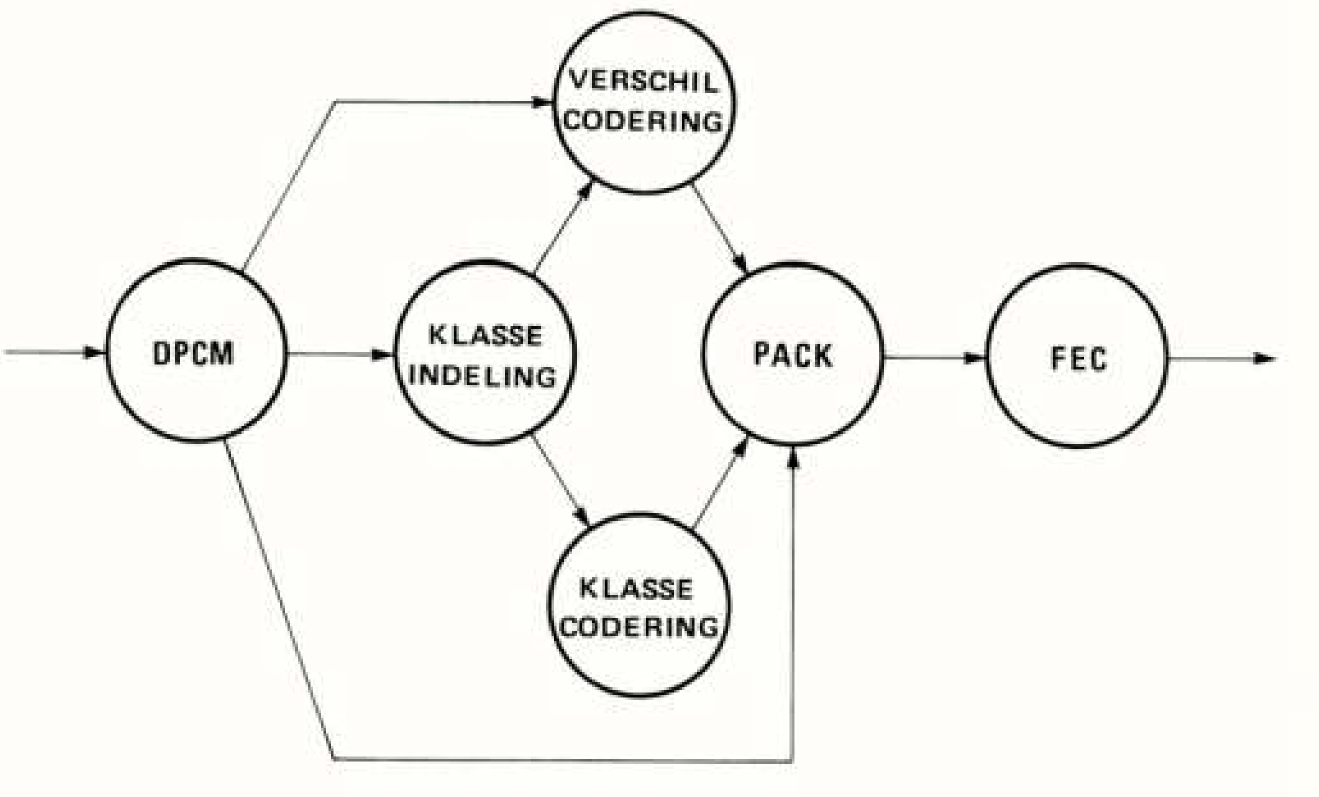


Fig. 7 MEANDER compressie algoritme

Als de doorvoersnelheid van het hele systeem nog verder verhoogd moet worden dan de nu vereiste 166.6 Kbit/s, moet er echter gezocht worden naar andere oplossingen. Mogelijkheden kunnen gevonden worden in:

- Verder onderzoek naar compressie algoritmen die mogelijk eenvoudiger parallelliseerbaar zijn, of die direct gebaseerd zijn op een parallel bewerking.

  Het probleem is namelijk dat bij het implementeren van het MEANDER algoritme geprobeerd wordt een in beginsel sequentieel algoritme in een parallelle structuur uit te voeren.
- Parallel uitvoeren van delen van het MEANDER-algoritme. Een probleem hierbij vormt de 'pack'-functie. Als dit parallel wordt uitgevoerd zal de gecomprimeerde beelddata niet meer op volgorde verstuurd worden. Dit vereist dan weer extra inspanning van de decompressiefunctie.

  Misschien is het nog mogelijk om een deel van de 'pack' functie, die veel schuifoperaties van lange reeksen bits over korte afstanden vereist, in hardware uit te voeren. Dit kan dan misschien in combinatie met de hardware die gebruikt wordt voor de FEC-functie.

- Gebruiken van de "packetized-telemetriestandaard" voor het versturen van data over het kanaal. De drie verschillende spectrale kanalen kunnen dan onafhankelijk gecomprimeerd worden en vervolgens op frame-basis worden samengevoegd.

6 Conclusies

Naar aanleiding van het gebruik van Transputers binnen het METEODIS project kunnen een aantal conclusies getrokken worden:

- De Transputer lijkt een geschikte component om als basis te dienen voor het maken van high-performance dedicated systemen. Problemen die met behulp van parallelle algoritmen kunnen worden opgelost, kunnen met dedicated Transputer systemen met zeer hoge snelheid verwerkt worden. Bij geschikte problemen kan het aantal Transputers naar wens worden uitgebreid om de vereiste prestaties halen, zonder dat dit leidt tot een bottle-neck in de communicatie.
- Voor algemene toepassingen is de Transputer nog moeilijk toegankelijk.
  Mogelijk wordt dit verbeterd door nieuwere besturingssystemen. Een mogelijk voorbeeld hiervan is het HELIOS-beheerssysteem, waar een 'pool' van Transputers naar de gebruiker / programmeur als een UNIX-achtig systeem gepresenteerd wordt.
- Occam biedt een mogelijkheid om parallelle algoritmen compact en duidelijk te beschrijven.
  Gebruikers raken ervan doordrongen dat binnen programma's sequentieel denken erg vanzelfsprekend is.
  Mogelijk bieden Occam en andere op parallel processing gerichte programmeertalen handvatten om te komen tot een directer parallel begrip van problemen.
- Het parallelliseren van beschikbare sequentiële algoritmen levert vaak grote problemen op.
  Als de oplossing van een probleem mogelijk op een parallel systeem moet worden uitgevoerd, dan zal dit direct bij de ontwikkeling van het algoritme moeten worden meegenomen.

7 Referenties

(1) The Transputer family 1988
Inmos Ltd.

(2) Parallel computers 2;
Architecture, programming and algorithms.
R.W. Hockney, C.R. Jesshope
1988; IOP publishing

(3) A tutorial introduction tot Occam programming
Dick Pountain Inmos Ltd 72 OCC 04307

(4) Occam: language definition
David May Inmos Ltd 72 OCC 04402

(5) METEODEC & METEOCRYPT
Volume II Functional specification
Börger 1988; NLR-TR 88156
Contract report: ESOC/ESTEC

(6) METEODEC & METEOCRYPT
Volume III Architectural design
Börger 1989; NLR-TR 88156
Contract report: ESOC/ESTEC

(7) Implementation aspects of a decompressor for compressed METEOSAT data on a VME/68000 system.
1987; NLR-TR 87003 L
M.H.J.B. Versteeg

Voordracht gehouden tijdens de 369e werkvergadering.

UIT HET NERG

LEDENMUTATIES

Voorgestelde leden

Ir. D. Beaufort, Watersteeg 4d, 2311 HZ LEIDEN.

Ing. A.C. van der Jagt, Bugel 48,
2907 GA CAPELLE A/D IJSSEL.

Drs. P.F.J. van Velthoven, Molenwerfsteeg 27,
3514 BZ UTRECHT.

Mw. ing. E. Harreveld, V. Boetzelaerlaan 47,
2581 AB 's-GRAVENHAGE.

Nieuwe leden

Ir. P.F.C. Blankers, Elviraland 314,
2591 GR 's-GRAVENHAGE.

Ir. E.W. Bol, Melis Stokezijde 170,
2543 GK 's-GRAVENHAGE.

Ir. J. Hof, Vaartweg 17^{I}, 1211 JD HILVERSUM.

Ir. A.D.A. Massar, Montgomerylaan 178, 2625 PT DELFT.

Ir. P.R.J.M. Smits, Jul. van Stolberglaan 206,
2595 CM 's-GRAVENHAGE.

Ir. J.C. Stekelenburg, Vlakkeeweg 6, 7051 GH VARSSEVELD.

Nieuwe adressen van leden

Ir. A.G. van Lienden, Hendersonstraat 173,
2286 XA RIJSWIJK.

Ir. J. Elshout, Oud Milligenseweg 47,
3886 MD GARDEREN.

Ir. B.J.A. Koenderink, Singraven 33, 7608 BA ALMELO.

Ir. D.A. van der Meij, Ruusbroecstraat 11,
8022 EA ZWOLLE.

Ing. D. Oorburg, Van Beethovenlaan 16,
2394 HC HAZERSWOUDE.

Ir. Tj.J. Tjalkens, Fuchsiastraat 29, 5644 LP EINDHOVEN.

Ir. Y.C.M. van der Werf, Homeruslaan 20, ZEIST.

Ir. H.C. van der Sluis, Nobellaan 2, 2641 XP PIJNACKER.



IR. H. A. M. LUIJF

NEDERLANDS ELEKTRONICA- EN RADIOGENOOTSCHAP
(369ste werkvergadering)
IEEE BENELUX SECTIE
AFDELING TELECOMMUNICATIE KIVI

UITNODIGING

voor de lezingendag op **woensdag 7 juni 1989** op het **Fysisch en Electronisch Laboratorium TNO (FEL-TNO), Oude Waalsdorperweg 63, Den Haag.**
THEMA : **"MASSIEF REKENEN".**



IR. A. C. M. OERLEMANS

PROGRAMMA :

9.45 uur: Ontvangst en koffie.

10.15 uur: **IR. H. A. M. LUIJF,** (FEL-TNO, Den Haag);
"INTRODUCTIE TOT MASSIEF REKENEN EN DE INFRASTRUCTUUR DAAR OMHEEN; EEN FEL-TNO VISIE."

11.00 uur: **IR. G. J. HAMEETMAN,** (NLR, Amsterdam);
"DE NEC SX-2 SUPERCOMPUTER BIJ HET NLR: ERVARINGEN NA 1 JAAR GROOTSCHALIG REKENWERK."

11.45 uur: Koffie.

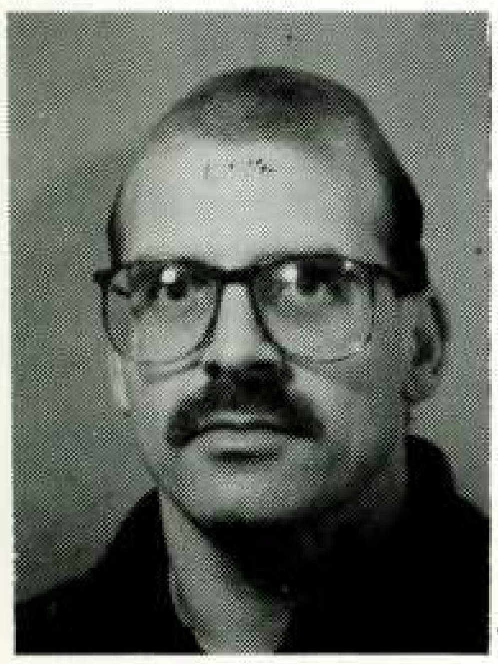
12.15 uur: **IR. A. C. M. OERLEMANS,** (Philips Nat. Lab., Eindhoven);
"PARALLEL PROCESSING BIJ PHILIPS."

13.00 uur: Lunch.

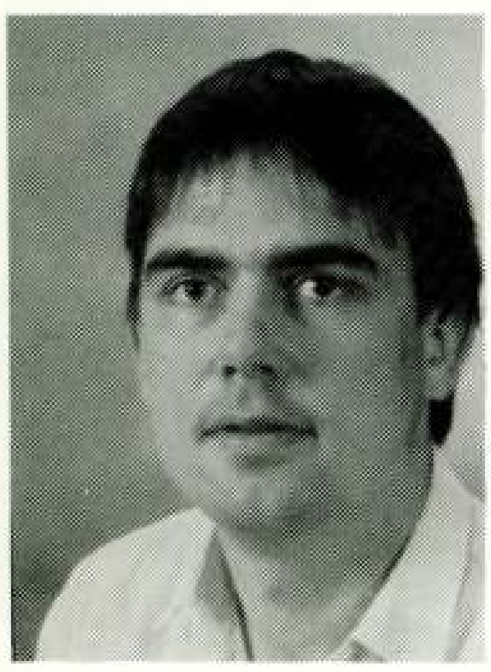
14.00 uur: **IR. P. L. J. VAN LIESHOUT,** (FEL-TNO, Den Haag);
"VISUALISATIE VAN 3D EMPIRISCHE DATA: DE VOXEL PROCESSOR."

14.45 uur: Thee.

15.15 uur: **IR. M. H. J. B. VERSTEEG,** (NLR, Amsterdam);
"TRANSPUTERS, PARALLEL REKENEN IN EMBEDDED SYSTEMEN T.B.V. BEELDCOMPRESSIE VOOR RUIMTEVAARTTOEPASSINGEN."



Aanmelding voor de lezingen dient te geschieden vóór 27 mei d.m.v. de aangehechte kaart **gefrankeerd met 55 cent.** Lunch reservering vindt slechts plaats indien **vóór 27 mei** een bedrag van ***f* 15,00** is ontvangen op postrekening 94746 t.n.v. Penningmeester NERG, Leidschendam, onder vermelding van "Massief rekenen". Leden van NERG, IEEE en KIVI en studenten hebben gratis toegang tot de lezingen. Tevens kunnen studenten de helft van de vervoerskosten vergoed krijgen (openbaar vervoer (2e kl) of anders, mits goedkoper).
Niet-leden dienen een entree-prijs van *f* 15,00 te betalen. Deelnemers dienen de uitnodigingskaart mee te nemen en op verzoek te tonen bij de toegang tot terrein en gebouw.
Het Fysisch en Electronisch Laboratorium TNO is te bereiken als aangegeven op bijgaand kaartje.



IR. M. H. J. B. VERSTEEG

Namens de samenwerkende verenigingen,
DR. G. W. M. VAN MIERLO, NERG.
Tel. 070 - 264221, tst 307.

Den Haag, mei 1989.

Conferentieaankondigingen

Symposium Telematica op 25-10-1989 wordt georganiseerd door de Fysisch-Mathematische Faculteitvereniging van de RUG.. Aanvang 10.30 uur, einde 17.00 uur. Nadere informatie: tel. 050-634947.

PATO cursussen
Digitale signaalverwerking op 13, 14, 20, 21, 27 en 28-11-1989, TU Eindhoven.
Glasvezelcommunicatie op 15, 22 en 29-11-1989, TU Eindhoven.
Elektro-magnetische compatibiliteit op 16, 17, 23, 24 en 30-11-1989, TU Eindhoven.
Nadere informatie: Bureau Pato orgaan, Postbus 30424, 2500 GK 's Gravenhage, tel. 070-644957.

Vierde cursus Optika en Lasers wordt door Optel georganiseerd op de vrijdagen van 20-10-1989 tot 19-01-1990 in de Nijmeegse universiteit. Nadere informatie: Mw. G. van Heugten, tel. 080-613111.

Inhoud

deel 54 – nr. 3 – 1989

Druk: de Witte, Eindhoven.